Induction interactive d'extracteurs n-aires pour les documents semi-structurés

Patrick Marty

Induction interactive d'extracteurs n-aires pour les documents semi-structurés

THÈSE

présentée et soutenue publiquement le 4 Décembre 2007

pour l'obtention du

Doctorat de l'Université Charles de Gaulle - Lille III
Mention Informatique

par

Patrick Marty

Composition du jury

Rapporteurs	Boris Chidlovskii, Chercheur	Xerox Research Centre Europe
	Patrick Gallinari, Professeur	Université Paris VI
Examinateurs	Patrice Buche, Ingénieur de Recherche	INRA HDR
	Marc Tommasi, Maître de Conférence	Université Lille III
	Fabien Torre, Maître de Conférence	Université Lille III
Directeur	Rémi Gilleron, Professeur	Université Lille III

UNIVERSITÉ LILLE III
Groupe de Recherche sur l'APPrentissage Automatique

On ne regarde plus les étoiles, mais les écrans
Paul Virilio

Remerciements

Je remercie Rémi Gilleron, Professeur, qui a dirigé ma thèse, ainsi que Marc Tommasi et Fabien Torre qui m'ont co-encadré.Merci pour leurs conseils, leur recul et leur capacité à toujours remettre leur travail scientifique en question, dans le but de l'améliorer.

Je remercie également Patrick Gallinari et Boris Chidlovskii d'avoir bien voulu accepter la charge de rapporteur.

Je remercie tout particulièrement Patrice Buche d'avoir bien voulu juger ce travail.

Je remercie aussi toute l'équipe du GRAppA et celle du projet Mostrare. Merci à tous ceux qui ont contribué à la bonne ambiance de l'ex-petite maison, du loft et de la Haute-Borne. Les déménagements successifs dans de nouveaux locaux n'ont jamais modifié la bonne entente au sein de ces deux équipes, ainsi que l'ambiance chaleureuse qui y règne (à part peut-être les pannes des radiateurs du loft).

Un grand merci à Hahn-Missi Tran et Matthieu Keith pour leur contribution brillante au développement de la plate-forme d'extraction miele, dans laquelle s'intègrent les algorithmes proposés dans cette thèse.

Et bien sûr, je n'oublie pas Dominique Gonzalez pour ses qualités d'enseignant et de pédagogie, et les signatures de ses mails. Merci à Alain Terlutte pour sa rigueur scientifique et ses fabuleux automates à os (et à pas). Merci à Isabelle Tellier pour le soutien qu'elle m'a apporté quand j'en ai eu besoin, pour sa culture cinématographique qu'elle partage avec enthousiasme et pour sa maladresse légendaire ;-). Je remercie également Jérémie Mary pour ses encouragements et sa vision des choses, Aurélien Lemay pour ses explications sur les automates d'arbres, Francesco De Comité pour les modifications de C4.5, Yves André pour `XQUERY`, et tous les ex-thèsards de l'équipe qui ont enrichi mon quotidien depuis mon arrivée au GRAppA.

Enfin je remercie très chaleureusement tous ceux qui m'ont apporté leur soutien moral tout au long de mes années de thésard.

Table des matières

Table des matières **1**

1 Introduction **5**
1.1 Contexte . . . 5
1.2 Extraction d'information . . . 6
1.2.1 Extraction depuis les documents non structurés . . . 7
1.2.2 Extraction depuis les documents semi-structurés . . . 8
1.2.2.1 Documents `HTML` . . . 11
1.2.2.2 Documents `XML` . . . 11
1.2.3 Bilan . . . 12
1.3 Tâches d'extraction . . . 12
1.3.1 Entrée d'une tâche d'extraction d'information . . . 12
1.3.2 Nature de la sortie . . . 17
1.3.3 Bilan . . . 20
1.4 Conception d'extracteurs . . . 20
1.4.1 Écriture manuelle . . . 20
1.4.2 Spécification assistée . . . 23
1.4.3 Induction supervisée d'extracteurs . . . 25
1.4.4 Induction non supervisée . . . 25
1.4.5 Bilan . . . 28
1.5 Approche supervisée pour l'induction d'extracteurs . . . 28
1.6 Objectifs et contribution . . . 31
1.7 Plan de la thèse . . . 32

2 Induction d'extracteurs et classification supervisée **33**
2.1 Introduction . . . 33
2.2 Classification Supervisée . . . 33
2.2.1 Généralités . . . 33
2.2.2 Représentation en attribut valeur . . . 35
2.2.3 Quelques algorithmes d'apprentissage . . . 35
2.2.3.1 Arbres de décision . . . 36
2.2.3.2 Apprenants à base de moindres généralisés . . . 37
2.3 La Classification Supervisée appliquée à l'Extraction d'Information . . . 39
2.3.1 BWI . . . 40
2.3.1.1 Représentation des documents . . . 41

2.3.1.2 Définition et codage des exemples 41
2.3.1.3 Langage d'hypothèses 41
2.3.1.4 Choix des exemples positifs et négatifs 41
2.3.1.5 Algorithme d'extraction 42
2.3.1.6 Algorithme d'apprentissage 42
2.3.2 ELIE 43
2.3.2.1 Représentation des documents 43
2.3.2.2 Définition et codage des exemples 43
2.3.2.3 Langage d'hypothèses 44
2.3.2.4 Algorithme d'extraction 44
2.3.2.5 Choix des exemples positifs et négatifs 44
2.3.2.6 Algorithme d'apprentissage 44
2.3.3 PAF$_{texte}$ 45
2.3.3.1 Représentation des documents 45
2.3.3.2 Définition et codage des exemples 46
2.3.3.3 Langage d'hypothèses 47
2.3.3.4 Algorithme d'extraction 47
2.3.3.5 Choix des exemples positifs et négatifs 47
2.3.3.6 Algorithme d'apprentissage 47
2.4 Approche générique 48
2.4.1 Choix de représentations 48
2.4.1.1 Représentation des documents 48
2.4.1.2 Nature et codage des exemples 48
2.4.2 Algorithme générique d'extraction 49
2.4.3 Algorithme générique d'induction d'extracteur 50
2.5 Conclusion 51

3 Extraction n-aire dans les structures arborescentes 53
3.1 Introduction 53
3.2 Différentes tâches d'extraction n-aire 53
3.2.1 Répartition des n-uplets 54
3.2.2 Granularité d'une composante 54
3.2.2.1 Extraction d'une feuille 54
3.2.2.2 Extraction d'une partie du contenu d'une feuille 54
3.2.2.3 Extraction de plusieurs feuilles 55
3.2.3 Bilan 55
3.3 Stockage de n-uplets dans un arbre 55
3.3.1 Organisation en table 56
3.3.2 Organisation en liste 58
3.3.3 Organisation en table tournée 60
3.3.4 Organisation avec factorisation 63
3.3.5 Organisation en table croisée 65
3.3.6 Caractéristiques 68
3.3.7 Bilan 68
3.4 Expressivité des systèmes majeurs d'induction d'extracteurs 69
3.4.1 WIEN 70

3.4.2 SOFTMEALY . . . 73
3.4.3 STALKER . . . 73
3.4.4 LIPX . . . 74
3.4.5 SQUIRREL$_n$ . . . 74
3.5 Conclusion . . . 75

4 Induction d'extracteurs n-aire **77**
4.1 Introduction . . . 77
4.2 Cadre de travail . . . 78
4.3 Aperçu de l'approche . . . 78
4.4 Représentation des documents . . . 79
4.5 Représentation des n-uplets . . . 81
4.5.1 Codage d'un nœud . . . 82
4.5.2 Codage d'une feuille . . . 82
4.5.3 Codage d'une dépendance entre deux feuilles . . . 82
4.5.4 Codage d'un tuple . . . 86
4.5.5 Complexité du codage . . . 88
4.6 Algorithme d'extraction . . . 88
4.6.1 Algorithme . . . 88
4.6.2 Complexité . . . 90
4.6.3 Propriétés . . . 90
4.6.3.1 Valeurs manquantes . . . 91
4.6.3.2 Ordre d'extraction pour les organisations de base . . . 92
4.6.3.3 Expressivité en terme de requêtes n-aires . . . 92
4.7 Algorithme d'induction . . . 93
4.7.1 Algorithme . . . 93
4.7.2 Consistance . . . 94
4.7.3 Implémentation . . . 95
4.8 Expériences . . . 96
4.8.1 Évaluation . . . 96
4.8.1.1 Correction des données extraites dans le cas unaire . . . 97
4.8.1.2 Correction des données extraites dans le cas n-aire . . . 97
4.8.1.3 Précision, Rappel et F-mesure . . . 97
4.8.1.4 Couverture . . . 98
4.8.1.5 Protocole expérimental . . . 99
4.8.2 Expériences sur le corpus DataFoot . . . 99
4.8.2.1 Description du corpus . . . 99
4.8.2.2 Impact de l'algorithme de classification supervisée . . . 100
4.8.2.3 Valeurs manquantes . . . 101
4.8.3 Expériences sur le corpus Rise . . . 102
4.8.3.1 Description du corpus . . . 103
4.8.3.2 Résultats . . . 103
4.8.4 Expériences sur le corpus CorporateData . . . 105
4.8.4.1 Description du corpus . . . 105
4.8.4.2 Résultats . . . 105
4.9 Conclusion . . . 106

5 Induction interactive d'extracteurs **109**
5.1 Introduction . . . 109
5.2 Cadre de travail . . . 110
5.2.1 Annotation partielle . . . 110
5.2.2 Modélisation de l'utilisateur . . . 111
5.2.2.1 Hypothèses faites sur l'utilisateur . . . 111
5.2.2.2 Actions possibles pour l'utilisateur . . . 111
5.2.3 Scénario interactif . . . 112
5.3 Algorithmes interactifs d'induction . . . 113
5.3.1 Algorithme interactif version 1 . . . 114
5.3.2 Algorithme interactif version 2 . . . 115
5.3.3 Algorithme interactif version 3 . . . 116
5.4 Apprentissage d'un classificateur c_i . . . 116
5.4.1 Fonction $ApprendreClassificateur$ . . . 117
5.4.2 Sélection des exemples négatifs pour le document courant . . . 118
5.4.2.1 Notion de chemin étendu . . . 119
5.4.2.2 Moindre généralisé de chemins étendus . . . 119
5.4.2.3 Fonction Negp . . . 121
5.5 Expériences . . . 122
5.5.1 Modélisation de l'utilisateur . . . 122
5.5.2 Protocole expérimental . . . 123
5.5.3 Résultats . . . 124
5.6 Conclusion . . . 124

Conclusion **127**

A Annexes : Quelques systèmes d'induction supervisée d'extracteurs **129**
A.1 WIEN . . . 130
A.2 SOFTMEALY . . . 132
A.3 STALKER . . . 134
A.4 LIPX . . . 136
A.5 SQUIRREL$_n$ . . . 137
A.6 Bilan . . . 138

Bibliographie **144**

Table des figures **145**

Chapitre 1

Introduction

Extraire : tirer avec une certaine difficulté une chose de ce qui la contient
Dictionnaire Hachette – 1988

1.1 Contexte

Depuis son invention en 1989 par Tim Berners-Lee, le Web propose un nombre sans cesse croissant de sources d'informations. L'exploitation des données de ces sources très hétérogènes est devenue, ces dernières années, une problématique majeure pour de nombreux chercheurs. Cette tendance est accentuée par le développement du *Web sémantique*. Le Web sémantique est une évolution du Web classique dont le but est de rendre le contenu du Web accessible et utilisable par des agents logiciels grâce à des annotations sémantiques [5]. Ainsi ces agents auront la possibilité de réaliser des traitements, actuellement effectuées essentiellement par des humains, comme la recherche, le partage et l'intégration d'informations. Par exemple, un agent serait capable de chercher sur le Web la liste des livres d'un auteur donné qui sont disponibles dans la librairie la plus proche et qui est ouverte jusqu'à 19h30 le samedi soir. Actuellement seul un humain peut réaliser une telle tâche, pas une machine. En effet il est nécessaire d'utiliser plusieurs moteurs de recherches (celui pour trouver les librairies proches, celui de la librairie pour trouver la listes des livres recherchées,...) et de combiner leurs résultats. Le but du Web sémantique est de rendre possible un tel traitement par une machine.

Même si de plus en plus de chaînes de publication et de diffusion de données sur le Web, comme les systèmes de gestion de contenu, intègrent un processus d'annotation, les informations sémantiques font cruellement défaut au Web actuel. Pour pallier à ces manques, différents travaux de recherche se sont focalisés sur l'annotation sémantique automatique des données du Web [26, 86, 79]. Une composante de cette tâche ardue et ambitieuse est l'acquisition des connaissances du contenu du Web. D'après [7] ce processus se divise conceptuellement en trois étapes successives :

1. la *recherche d'information* dont le but est de découvrir des documents contenant des données pertinentes par rapport à une requête ;
2. l'*extraction d'information* qui consiste à extraire des informations pertinentes

de ces documents ;

3. et enfin l'*intégration d'information* dont le propos est de fusionner les données obtenues à partir de différentes sources hétérogènes du Web en une base homogène de connaissances.

Cette thèse se focalise sur le second point et plus particulièrement sur la tâche d'extraction d'information depuis les documents du Web.

Avec l'évolution du Web, les travaux de recherche en extraction d'information ont progressivement évolués des documents non-structurés, comme les mails, vers les documents semi-structurés, comme par exemple les pages `HTML`. Actuellement les documents semi-structurés sont de plus en plus présents, notamment avec l'essor du format `XML` (*eXtensible Markup Language*) qui s'impose comme le langage universel d'échange et de diffusion d'informations. Un document `XML` est constitué d'éléments organisés de manière arborescente. Chaque élément est représenté par une paire de balise ouvrante/fermante et l'imbrication des balises décrit la structure d'arbre d'un document `XML`.

On distingue deux cas typiques d'utilisation d'`XML` :

- comme format de description/diffusion de documents, comme par exemple `XHTML` (la version `XML` de `HTML` pour la description des pages Web) et le format de documents bureautique `ODF`, dans ce cas on dit que `XML` est orienté document ;
- comme format de description/diffusion de données, comme par exemple `RDF` et le format `SOAP` d'envoi de messages entre applications distantes, dans ce cas on dit que `XML` est orienté donnée.

Malgré ces deux orientations distinctes, le besoin commun d'extraire de l'information apparaît. Dans le cas des documents `XML` orientés document il peut s'agir d'extraire des métadonnées contenues dans les documents, comme la liste des rédacteurs et des mots clés. Pour les documents `XML` orientés donnée, on considère par exemple l'extraction de certains paramètres et valeurs d'un message `SOAP`. Le développement grandissant des Web services et l'usage considérable qu'ils font d'`XML` est un argument supplémentaire pour justifier le besoin d'extraire de l'information des documents semi-structurés.

Après avoir esquisser le contexte dans lequel se place cette thèse, nous allons tenter de définir plus précisément la notion d'extraction d'information.

1.2 Extraction d'information

Dans la littérature scientifique on rencontre plusieurs définitions de l'*extraction d'information* . Pour [23], elle consiste à extraire des faits et de la connaissance de documents. [67] définit l'extraction d'information par comparaison à la recherche d'information : la recherche d'information consiste à trouver un ensemble de documents pertinents, tandis que l'extraction d'information consiste à trouver dans ces documents un ensemble de faits pertinents. Ces deux définitions sont très générales. Elles ne spécifient ni le type des documents d'entrée, ni la nature des éléments extraits. [28] propose une définition plus précise et focalisée sur le Web. Les pages d'un site Web sont considérées comme des conteneurs de données. L'extraction d'information consiste alors à produire une représentation structurée de ces données. La définition de [19] est également

spécifique au Web et base l'extraction d'information sur l'usage d'*extracteurs*. Un extracteur est défini comme un programme qui en accédant à un site Web lui permet de se comporter comme une base de données. Ainsi il est possible d'interroger les données contenues dans les pages d'un site comme si elles se trouvaient stockées dans une base de données. [12] va dans le même sens en considérant un extracteur comme un programme posant une requête sur des documents semi-structurés, comme les documents `HTML` ou `XML`. Les requêtes envisagées par [12] permettent d'extraire des données selon des contraintes sur la structure des documents. Enfin pour [44] l'extraction d'information est un cas particulier de transformation d'arbres, avec comme entrée un document semi-structuré et comme sortie l'arbre contenant les données extraites.

Malgré les différences de ces quelques définitions, on peut en dégager les caractéristiques suivantes :

- l'extraction d'information a comme entrée un ensemble de documents d'un certain type ;
- ces documents contiennent des informations ou données ;
- la sortie d'une tâche d'extraction d'information est un ensemble de données structurées.

Dans cette thèse, nous proposons une définition de l'extraction d'information selon son entrée et sa sortie. Nous considérons que l'*extraction d'information* consiste à produire automatiquement des informations structurées à partir d'un ensemble de documents. En pratique une tâche d'extraction d'information est réalisée par un programme nommé *extracteur* que l'on peut définir comme une fonction de l'espace des documents d'entrée vers l'ensemble des structures de sortie. Un extracteur peut également être vu comme une requète sur les documents d'entrée.

Les différentes tâches d'extraction d'information possibles, et que l'on rencontre dans la littérature, se déclinent selon l'entrée et la sortie considérées. La nature des documents d'entrée est diverse :

- textes en langage naturel comme les dépêches de presse, les courriers électroniques, les textes de lois ;
- documents semi-structurés comme les documents `XML` ou les pages Web (`XHTML`).

De plus l'ensemble des documents considérés peut être homogène. C'est par exemple le cas d'un corpus d'actes de ventes d'un cabinet de notaires, tous formatés de la même manière. Les documents peuvent au contraire être très hétérogènes. Il s'agit, par exemple, de l'ensemble des pages Web des sites de météorologie accessibles sur Internet, chaque site ayant une mise en page spécifique de l'information.

Selon le type de documents d'entrée, différentes approches ont été proposées. On distingue d'une part celles travaillant sur les *documents non-structurés*, comme les textes en langue naturelle, et d'autre part celles portant sur les *documents semi-structurés* comme par exemple les pages Web et les documents `XML`. Les deux sections suivantes abordent l'extraction dans les documents non-structurés puis dans les documents semi-structurés.

1.2.1 Extraction depuis les documents non structurés

Les conférences MUC, qui se sont déroulées de 1987 à 1997, se sont focalisées sur la compréhension de textes en langue naturel. Cette tâche est ramenée à l'extraction

d'information à l'aide de techniques de traitement automatique de la langue (TAL).

Considérer un texte en langage naturel comme un document non-structuré peut paraître choquant. En effet, un texte possède une structure grammaticale, que l'on peut exhiber à l'aide de techniques TAL. Mais cette structure n'est pas celle des données contenues dans un texte, même si elle peut aider à découvrir les informations à extraire. La notion de document non-structuré est à comprendre du point de vue des bases de données. Dans une base de données, la structure des données est connue à travers le schéma de la base, ce qui rend aisé leur manipulation et leur interrogation par des requêtes. Dans ce cas on parle de données structurées. À l'opposé de cela on trouve les informations disséminées dans les textes dont la structure est *a priori* inconnue et très variable. Ici on ne dispose pas d'un schéma, qui indiquerait le type des données et leur organisation, pour interroger directement les données. C'est pourquoi on parle alors de documents non structurés, que l'on désigne également par l'expression *texte plat*.

Les tâches considérées dans les conférences MUC sont, par exemple, l'extraction depuis des récits d'attentats en Amérique du Sud d'informations comme la date, le lieu et le type d'attentat ou encore le nom et le type de l'organisation revendiquant l'attentat. La figure 1.1 présente un des textes considéres par MUC 4, tandis que la figure 1.2 illustre les informations à extraire de ce texte. Elles sont structurées sous la forme d'un enregistrement avec plusieurs champs.

Résoudre de telles tâches d'extraction nécessite à la fois des ressources, par exemple des dictionnaires pour identifier les entités nommées comme les noms de villes ou de pays, une part importante de connaissances linguistiques ainsi qu'une analyse du sens du texte.

1.2.2 Extraction depuis les documents semi-structurés

Alors que les conférences MUC se terminent, émergent des travaux sur l'extraction d'information depuis le contenu du Web et notamment les pages `HTML` avec une approche uniquement syntaxique [51, 42]. Ils sont basés sur l'exploitation des régularités syntaxiques de la structure des documents `HTML` et ne requièrent pas la machinerie linguistique des méthodes à base de traitement automatique de la langue naturelle.

Comme l'ont montré [2, 9, 28] et [1], les données du Web ne sont pas structurées comme celles des bases de données. Les documents du Web, comme les pages `HTML` et les documents `XML`, sont des *documents semi-structurés*. Comme pour les documents non-structurés, la notion de semi-structuré est à comprendre du point de vue des bases de données. Malgré la structure des documents, celle-ci n'est pas comparable à celle d'une base de données : elle est moins régulière, rigide et typée.

Les documents semi-structurés se décrivent eux-mêmes. Cette description est portée par le balisage des documents et le nom des balises, qui sont chargées de sémantique, surtout dans le cas des documents `XML`. Cette caractéristique rend les documents semi-structurés compréhensibles par un humain, même si leur usage est plus orienté vers le stockage, l'échange d'information et le traitement par programme. Les documents semi-structurés ont une structure arborescente, traduite par l'imbrication des balises, qui décrit à la fois structure logique du document et son contenu. Les nœuds internes de l'arbre représentant un document semi-structuré correspondent aux éléments de la

BOGOTA, 9 JAN 90 (EFE) -- [TEXT] RICARDO ALFONSO CASTELLAR, MAYOR OF ACHI, IN THE NORTHERN DEPARTMENT OF BOLIVAR, WHO WAS KIDNAPPED ON 5 JANUARY, APPARENTLY BY ARMY OF NATIONAL LIBERATION (ELN) GUERRILLAS, WAS FOUND DEAD TODAY, ACCORDING TO AUTHORITIES.

CASTELLAR WAS KIDNAPPED ON 5 JANUARY ON THE OUTSKIRTS OF ACHI, ABOUT 850 KM NORTH OF BOGOTA, BY A GROUP OF ARMED MEN, WHO FORCED HIM TO ACCOMPANY THEM TO AN UNDISCLOSED LOCATION.

POLICE SOURCES IN CARTAGENA REPORTED THAT CASTELLAR'S BODY SHOWED SIGNS OF TORTURE AND SEVERAL BULLET WOUNDS.

CASTELLAR WAS KIDNAPPED BY ELN GUERRILLAS WHILE HE WAS TRAVELING IN A BOAT DOWN THE CAUCA RIVER TO THE TENCHE AREA, A REGION WITHIN HIS JURISDICTION.

IN CARTAGENA IT WAS REPORTED THAT CASTELLAR FACED A "REVOLUTIONARY TRIAL" BY THE ELN AND THAT HE WAS FOUND GUILTY AND EXECUTED.

CASTELLAR IS THE SECOND MAYOR THAT HAS BEEN MURDERED IN COLOMBIA IN THE LAST 3 DAYS.

ON 5 JANUARY, CARLOS JULIO TORRADO, MAYOR OF ABREGO IN THE NORTHEASTERN DEPARTMENT OF SANTANDER, WAS KILLED APPARENTLY BY ANOTHER GUERILLA COLUMN, ALSO BELONGING TO THE ELN.

TORRADO'S SON, WILLIAM; GUSTAVO JACOME QUINTERO, THE DEPARTMENTAL GOVERNMENT SECRETARY; AND BODYGUARD JAIRO ORTEGA, WERE ALSO KILLED.

THE GROUP WAS TRAVELING IN A 4-WHEEL DRIVE VEHICLE BETWEEN CUCUTA AND THE RURAL AREA KNOWN AS CAMPANARIO WHEN THEIR VEHICLE WAS BLOWN UP BY FOUR EXPLOSIVE CHARGES THAT DETONATED ON THE HIGHWAY.

FIG. 1.1 – Un document non-structuré de MUC 4

0.	MESSAGE: ID	DEV-MUC3-0008 (NCCOSC)
1.	MESSAGE: TEMPLATE	1
2.	INCIDENT: DATE	05 JAN 90
3.	INCIDENT: LOCATION	COLOMBIA: BOLIVAR (DEPARTMENT): ACHI (TOWN)
4.	INCIDENT: TYPE	KIDNAPPING
5.	INCIDENT: STAGE OF EXECUTION	ACCOMPLISHED
6.	INCIDENT: INSTRUMENT ID	*
7.	INCIDENT: INSTRUMENT TYPE	*
8.	PERP: INCIDENT CATEGORY	TERRORIST ACT
9.	PERP: INDIVIDUAL ID	"GUERRILLAS" / "GROUP OF ARMED MEN" / "ARMED MEN"
10.	PERP: ORGANIZATION ID	"ARMY OF NATIONAL LIBERATION" / "ELN"
11.	PERP: ORGANIZATION CONFIDENCE	SUSPECTED OR ACCUSED / REPORTED AS FACT: "ARMY OF NATIONAL LIBERATION" / "ELN"
12.	PHYS TGT: ID	*
13.	PHYS TGT: TYPE	*
14.	PHYS TGT: NUMBER	*
15.	PHYS TGT: FOREIGN NATION	*
16.	PHYS TGT: EFFECT OF INCIDENT	*
17.	PHYS TGT: TOTAL NUMBER	*
18.	HUM TGT: NAME	"RICARDO ALFONSO CASTELLAR"
19.	HUM TGT: DESCRIPTION	"MAYOR OF ACHI": "RICARDO ALFONSO CASTELLAR"
20.	HUM TGT: TYPE	GOVERNMENT OFFICIAL: "RICARDO ALFONSO CASTELLAR"
21.	HUM TGT: NUMBER	1: "RICARDO ALFONSO CASTELLAR"
22.	HUM TGT: FOREIGN NATION	-
23.	HUM TGT: EFFECT OF INCIDENT	DEATH: "RICARDO ALFONSO CASTELLAR"
24.	HUM TGT: TOTAL NUMBER	-

FIG. 1.2 – Données à extraire du document de la figure 1.1

structure logique du document, tandis que le contenu du document est stocké dans les feuilles de l'arbre.

Nous distinguons deux types de documents semi-structurés :

– les documents HTML ;

– et les documents XML ;

que nous examinons dans les deux sections suivantes.

1.2.2.1 Documents HTML

Le langage HTML est le langage de mise en forme des pages Web. Il permet la création de documents plus riches que du texte plat, en décrivant à la fois la structure du document, son contenu et sa présentation (ou rendu). L'information des pages HTML est structurée par des paires de balises de mise en forme. Chaque paire est constituée d'une balise ouvrante, suite de caractères délimitée par les symboles < et >, et d'une balise fermante, suite de caractères délimitée par les symboles </ et >. Par exemple, dans le document HTML fictif de la figure 1.3, la balise h1 indique un titre et la balise b délimite du texte en gras.

```
<html>
<body>
<h1>Le mot du jour</h1>
<b>catachrèse</b> : figure de style qui consiste
à détourner un mot de son sens propre
</body>
</html>
```

FIG. 1.3 – Un exemple de document HTML

Les pages Web sont produites manuellement ou automatiquement par programme. Dans ce cas elles intègrent souvent des informations provenant d'une base de données, comme par exemple les pages de résultats d'un moteur de recherche ou les pages d'un site de commerce. Lors de la transformation des enregistrements de la base en un (ou plusieurs) document(s) HTML, la structure des données est soit perdue soit rendue implicite par le balisage de mise en forme. Ainsi on peut voir l'extraction d'information comme la transformation inverse (mais inconnue) de celle ayant produit les pages en question.

1.2.2.2 Documents XML

Depuis sa création en 1998 par le W3C, le format XML est devenu un standard pour l'échange et le stockage de données semi-structurés. Le nombre de dialectes XML disponibles en témoigne. On trouve, par exemple, XHTML la version XML de HTML, RDF qui permet de diffuser des méta-données sur le Web, SVG un langage de description de graphiques en mode vectoriel, MATHML un langage d'écriture de formules mathématiques, SMIL pour les objets multimédias, ODF et DOCBOOK pour l'édition de documents, EBXML

pour le commerce électronique, CML pour la chimie, WSDL qui est le langage de description des interfaces des services Web,*etc.*

La figure 1.4 présente un exemple de document XML. Un document XML est constitué d'éléments. Par exemple, le document de la figure 1.4 contient les éléments morceaux, morceau, titre, artiste, compositeur, genre. Chaque élément est représenté par une paire de balises ouvrante/fermante. Dans la figure 1.4, l'élément titre est représenté par les balises <titre> et </titre>. À la différence d'HTML où l'ensemble des balises est défini par une norme fixée, XML permet de choisir l'ensemble des balises utilisées. On peut ainsi créer son propre dialecte en fonction de ses besoins, en choisissant à la fois les balises et la sémantique qui leur est associée.

L'enchâssement des couples de balises représente la structure arborescente d'un document XML. Un document XML doit être bien formé : toute balise ouverte doit être fermée et les balises fermantes apparaissent dans l'ordre inverse des balises ouvrantes (cette contrainte n'est pas obligatoire dans le cas d'HTML). On peut contraindre la structure arborescente des documents XML à l'aide d'un schéma, comme une DTD ou un XML Schema, qui définit l'ensemble des balises possibles et la manière dont elles sont structurées.

Outre l'écriture manuelle, les moyens pour produire des documents sont divers : publication à l'aide de suites bureautique (comme par exemple OpenOffice) ou outils de reporting (comme JasperReports), transformation de pages HTML en XHTML, exportation de données pour l'échange et l'interopérabilité entre applications (comme par exemple la publication du contenu d'une base de données au format XML), *etc.*

1.2.3 Bilan

Le nombre de documents semi-structurés est grandissant et il est nécessaire de pouvoir les interroger efficacement. Étant donné l'orientation actuelle du Web, le développement de technologies qui reposent massivement sur le langage XML, comme par exemple les Web services, nous nous focalisons, dans cette thèse, sur l'extraction d'information depuis les documents semi-structurés, et particulièrement les pages XHTML.

1.3 Tâches d'extraction

Cette section illustre la diversité des tâches d'extraction selon deux critères :
- l'entrée, caractérisée par la représentation des documents et l'atomicité des données à extraire ;
- la nature de la sortie.

1.3.1 Entrée d'une tâche d'extraction d'information

Pour caractériser l'entrée d'un extracteur, on distingue deux aspects :
- la représentation des documents ;
- l'atomicité des valeurs à extraire ;

que nous examinons plus en détail dans ce qui suit.

```
<morceaux>
     <morceau>
         <titre>Calambre</titre>
         <artiste>La Academia</artiste>
         <compositeur>Astor Piazzolla</compositeur>
         <genre>Tango</genre>
     </morceau>
     <morceau>
         <titre>Luna (Full Moon Remix)</titre>
         <artiste>Alexki</artiste>
         <compositeur>Astor Piazzolla</compositeur>
         <genre>Tango Electro</genre>
     </morceau>
     <morceau>
         <titre>Tango Apasionado</titre>
         <artiste>Astor Piazzolla</artiste>
         <genre>Tango Nuevo</genre>
     </morceau>
     <morceau>
         <titre>Milonga de Mis Amores</titre>
         <artiste>D. Arienzo</artiste>
         <genre>Milonga</genre>
     </morceau>
     <morceau>
         <titre>Felino</titre>
         <artiste>Electrocutango</artiste>
         <genre>Tango Electro</genre>
     </morceau>
     <morceau>
         <titre>Diferente</titre>
         <artiste>Gotan Project</artiste>
         <genre>Tango Electro</genre>
     </morceau>
     <morceau>
         <titre>Caminito</titre>
         <artiste>Los Tubatango</artiste>
         <compositeur>Juan De Dios Filiberto</compositeur>
         <genre>Tango</genre>
     </morceau>
</morceaux>
```

FIG. 1.4 – Un exemple de document XML représentant une bibliothèque de musique

Représentation des documents semi-structurés Pour illustrer différentes manières de représenter un document semi-structuré, cette section se focalise sur les documents `HTML` et `XHTML`. Trois représentations sont abordées. Parmi elles, seule la première est spécifique aux documents `HTML`, les deux autres représentations s'appliquent également aux document `XML`.

En plus d'obtenir et de naviguer dans et entre les pages `HTML`, un navigateur Web permet également de les visualiser. Cette représentation des pages par leur *rendu*, *i.e.* leur aspect graphique et visuel, est connue de tout utilisateur d'Internet. Elle est obtenue à partir du code source d'une page `HTML`, qui définit la structure et le contenu de la page. Ainsi un document `HTML` est une *séquence* de symboles. Cette représentation est plus proche de la machine que la précédente. Une représentation plus abstraite est celle d'*arbre* déterminée par l'imbrication des balises `HTML`.

Représentation par le rendu La figure 1.5 illustre le rendu visuel dans un navigateur d'une page du site Web BLS. Cette représentation est essentiellement adaptée à la visualisation des pages par un être humain. Bien qu'elle soit utilisée par certains systèmes d'extraction d'information [43], nous considérons cette représentation comme la moins adaptée pour l'extraction d'information.

D'une part elle est d'une stabilité critiquable car les différences entre les moteurs de rendu sont notables. De nombreux tests[1] mettent clairement en évidence ces différences et illustrent la difficulté d'implémentation d'un moteur de rendu conforme aux normes. D'autre part les informations à partir desquelles le rendu d'un document est obtenu sont présentes dans le source `HTML` (ou accessibles grâce à lui, comme les informations des feuilles de styles `CSS`).

Représentation par une séquence Le rendu d'une page Web est obtenu à partir du code source de la page qui est une séquence de caractères. Cette représentation des documents sous la forme d'une séquence d'unités syntaxiques, ou *tokens*, est la plus courante en extraction d'information. L'atomicité d'un token varie d'un aspect purement syntaxique à un aspect plus sémantique :

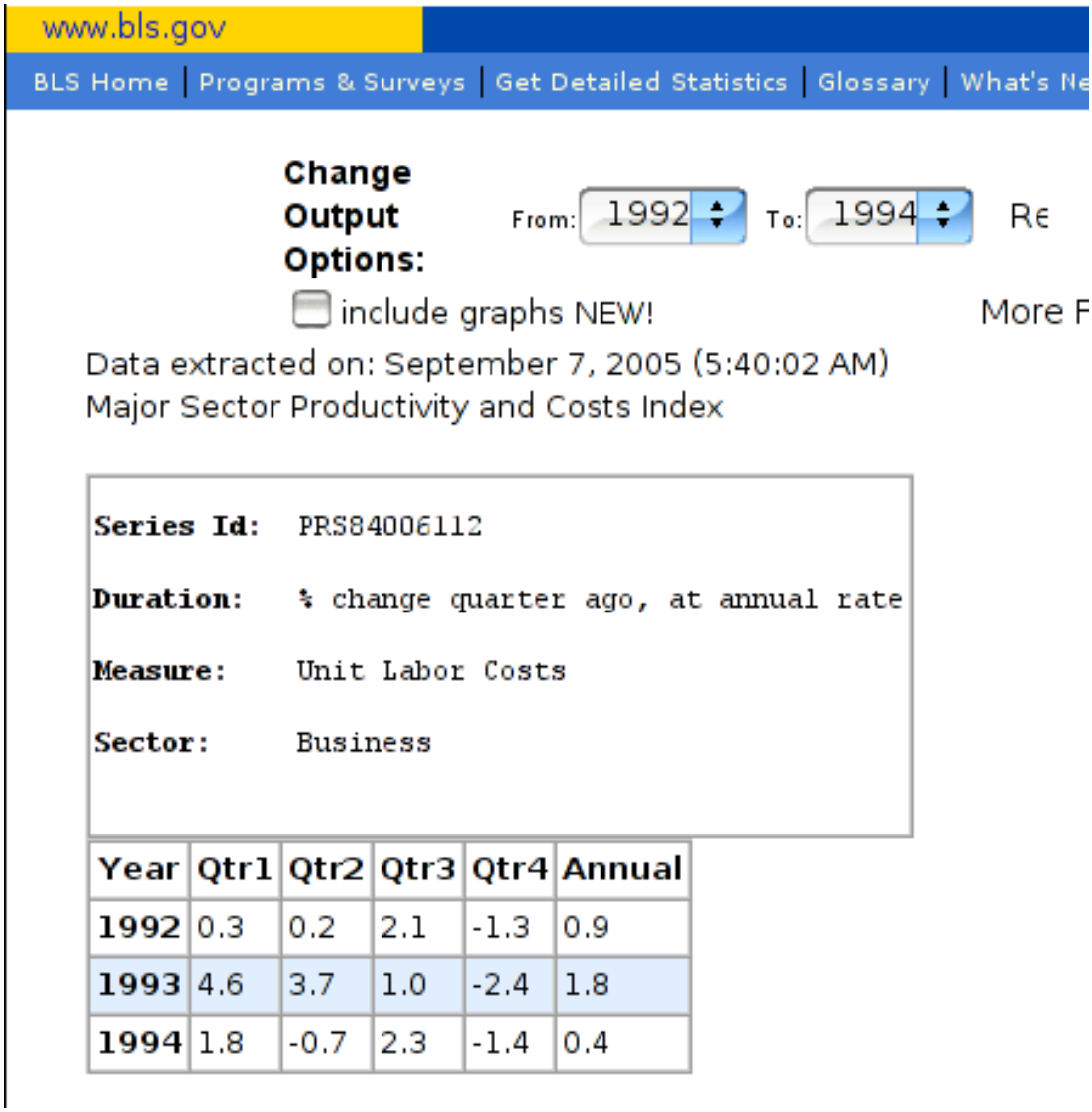
- caractère ;
- séquence de caractères définie par un critère syntaxique, comme par exemple une expression régulière ;
- unité sémantique plus complexe comme les mots définis à l'aide de dictionnaires ou d'outils de traitement du langage naturel (lèmmatiseur, identificateur d'entités nommées).

Dans le cas des documents semi-structurés un token peut être :

- une balise ouvrante ;
- une balise fermante ;
- toute autre séquence de caractères comprise entre deux caractères blancs (espace, tabulation, retour chariot).

L'inconvénient de cette vue est son expressivité limitée. Par exemple, dans le document de la figure 1.5 la valeur `Qtr1` est partagée par les triplets :
`(Qtr1,1992,0.3)`, `(Qtr1,1993,4.6)` et `(Qtr1,1994,1.8)`. Une observation du rendu

[1] comme le test Acid2 http://www.webstandards.org/files/acid2/test.html



Year	Qtr1	Qtr2	Qtr3	Qtr4	Annual
1992	0.3	0.2	2.1	-1.3	0.9
1993	4.6	3.7	1.0	-2.4	1.8
1994	1.8	-0.7	2.3	-1.4	0.4

FIG. 1.5 – Une partie d'une page Web du site BLS

```
<html> <body> <table> <tr> <th>Year </th> <th>Qtr1 </th> <th>Qtr2
</th> <th>Qtr3 </th> </tr> <tr> <th>1992 </th> <td>0.3 </td>
<td>0.2 </td> <td>2.1 </td> </tr> <tr> <th>1993 </th> <td>4.6
</td> <td>3.7 </td> <td>1.0 </td> </tr> <tr> <th>1994 </th>
<td>1.8 </td> <td>-0.7 </td> <td>2.3 </td> </tr> </table> </body>
</html>
```

FIG. 1.6 – Une partie du code source de la page Web de la figure 1.5

nous donne un critère pour associer la valeur `Qtr1` aux valeurs `0.3`, `4.6` et `1.8` : ces données sont dans une table et le trimestre est l'en-tête de la colonne qui contient les valeurs numériques. Exprimer cette propriété avec uniquement la représentation sous forme de séquence du document n'est pas possible. Cette impossibilité peut-être levée si l'on augmente la représentation séquentielle du document avec des informations provenant de la la sémantique des balises (`tr` est le constructeur d'une ligne et `td` d'une cellule de la table). Cependant une telle modification de la représentation est spécifique à la sémantique des balises. De plus on ne dispose pas toujours de cette dernière.

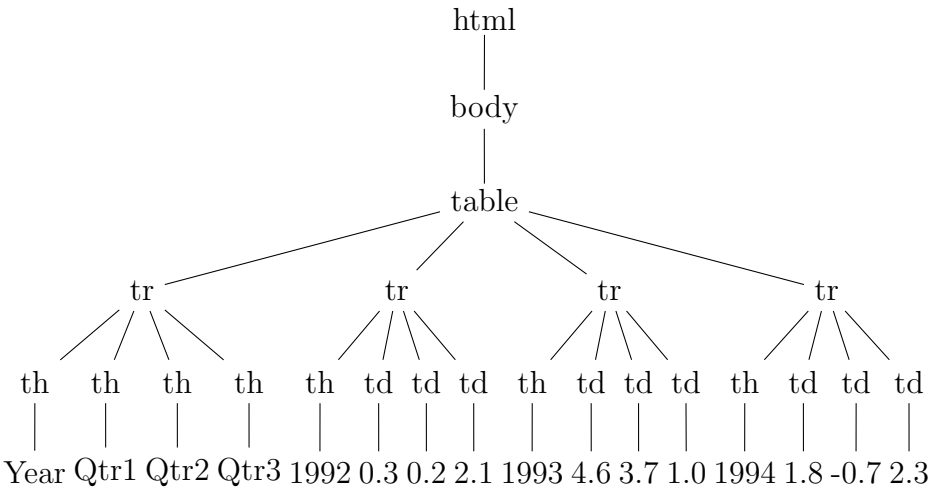


FIG. 1.7 – Représentation arborescente de la page Web de la figure 1.5

Représentation par un arbre Les documents à balises, comme `XML` et `HTML`, ont une nature arborescente intrinsèque. En effet, comme l'illustre la figure 1.7, l'imbrication des balises induit naturellement une structure d'arbre. Chaque paire de balises ouvrante/fermante définit un sous arbre dont la racine aura comme étiquette, ou label, le nom de la balise. Les portions de texte, qui ne sont pas des balises, sont les feuilles de l'arbre. Les données sont contenues dans les feuilles textes tandis que l'agencement des nœuds internes définit la structure des données.

Les arbres représentants les documents `XML` sont des arbres *à arité non bornée*. Un arbre d'arité non bornée, ou arbitraire, est un arbre dont le nombre de fils d'un nœud n'est pas borné. C'est le cas de l'arbre de la figure 1.7. Un autre exemple en `HTML` est celui de la balise `ul` qui annonce une liste non-numérotée. Chaque élément de la liste est désigné par la balise `li`. Dans l'arbre `HTML` chaque nœud `li` a comme père le nœud `ul`, constructeur de la liste. Une telle liste peut avoir un nombre quelconque d'éléments : le nombre de fils d'un nœud `ul` n'est pas borné.

Si l'on revient au document de la figure 1.5, sa représentation arborescente (figure 1.7) fournit un critère pour associer la valeur `Qtr1` aux valeurs `0.3`, `4.6` et `1.8` : le père de chacune de ces quatre feuilles textes se trouve à la même position parmi ses frères, à savoir en seconde position. Cette propriété se traduit sémantiquement par le fait que ces quatre valeurs sont dans la seconde colonne de la table. Cependant l'uti-

lisation de la sémantique des balises n'est pas nécessaire ici en raison d'une régularité structurelle que l'on découvre en observant l'arbre.

Atomicité des valeurs à extraire L'atomicité des valeurs à extraire dépend de la représentation choisie pour les documents. On se limite ici à la représentation séquentielle et la représentation arborescente.

Documents représentés comme une séquence Avec ce type de représentation, un document est un séquence de tokens et la valeur d'une composante à extraire est une sous-séquence de tokens [51, 42, 61]. Une valeur à extraire peut être :

- une partie d'un token ;
- un token dans son intégralité ;
- plusieurs tokens consécutifs.

Le choix des tokens joue ici un rôle capital. La plupart des travaux adaptent l'algorithme de tokenisation pour qu'une valeur à extraire correspondent soit à un token, soit à une séquence de tokens.

Documents représentés comme un arbre Lorsqu'un document est représenté par un arbre [81, 82, 56, 12, 36], les données à extraire se trouvent dans les feuilles de l'arbre. Dans ce cas, une donnée à extraire peut être :

- une portion d'une feuille ;
- exactement une feuille ;
- plusieurs feuilles consécutives.

Certains travaux [19, 81] considèrent chacune des feuilles textuelles de l'arbre comme une séquence de tokens qu'ils incorporent dans l'arbre. La feuille texte est remplacée par autant de feuilles qu'elle contient de tokens, à raison d'un token pour chaque nouvelle feuille créée.

1.3.2 Nature de la sortie

Il est nécessaire de distinguer trois sorties possibles pour une tâche d'extraction d'information :

- le cas unaire qui revient à extraire un ensemble de valeurs ;
- le cas n-aire qui consiste à extraire un ensemble de n-uplets de valeurs ;
- la sortie structurée sous forme d'arbre.

Nous allons illustré ces distinctions à l'aide du document `HTML` de la figure 1.8 qui présente à la fois le code source du document (en haut de la figure) et son rendu `HTML` (en bas).

Cas unaire La sortie d'une tâche d'extraction unaire est un ensemble de valeurs. Il s'agit d'extraire toutes les valeurs souhaitées et uniquement celles-ci. Ces valeurs sont de même type : elles correspondent à une même *composante*.

Par exemple, on peut extraire l'ensemble des noms de clubs de football du document `HTML` de la figure 1.8. Dans ce cas les valeurs à extraire sont `PSG`, `LOSC` et `OM`. On peut aussi extraire l'ensemble des scores, c'est à dire les valeurs `17` et `31`.

```
<html> <body> <table>
<tr><th>Club</th><td>PSG</td><td>LOSC</td><td>OM</td></tr>
<tr><th>Saison</th><td>2002</td><td>2004</td><td>2003</td></tr>
<tr><th>Score</th><td>17</td><td></td><td>31</td></tr>
</table> </body> </html>
```

Club	PSG	LOSC	OM
Saison	2002	2004	2003
Score	17		31

FIG. 1.8 – Un document semi-structuré : une page `HTML`, avec en haut son code source et en bas son rendu

Cas n-aire La sortie d'une tâche d'extraction n-aire est un ensemble de n-uplets de valeurs. Pour un n-uplet, les différentes valeurs sont *en relation* entre elles. Chacune d'elles correspond à une *composante*. L'ensemble des n-uplets à extraire est généralement désigné par une *relation cible*, qui exprime la relation entre les composantes. Les n-uplets à extraire sont des instances de la relation cible. L'arité de cette relation, *i.e.* la valeur de n, est le nombre de composantes qui la constituent. Par exemple, pour le document `HTML` de la figure 1.8 la relation cible est la relation ternaire `(Club,Saison,Score)`. Elle est constituée des composantes :

- `Club` qui est le score d'un club de football ;
- `Saison` qui désigne la saison considérée pour le club ;
- et `Score`, le score du club.

Les triplets à extraire sont `(PSG,2002,17)`,`(LOSC,2004,null)` et `(OM,2003,31)`. On constate que le triplet `(LOSC,2004,null)` comporte une valeur spéciale, notée `null`, qui représente l'absence de valeur pour la composante `Score` pour ce triplet. Il s'agit d'une *valeur manquante*.

Dans le cas n-aire, il ne suffit pas d'identifier correctement l'ensemble des valeurs de chaque composante. Il faut également que les n-uplets soient corrects, c'est à dire que pour chaque n-uplet extrait les valeurs de ses composantes soient bien liées entre elles. Si le triplet `(PSG,2004,31)` est extrait du document de la figure 1.8, alors l'extraction des composantes une à une est correcte : `PSG` est bien le nom d'un club, `2004` est bien une saison et `31` est bien le score d'un club. Cependant, ce couple n'est pas correct car l'association entre les trois composantes est erronée : `31` n'est pas le score du `PSG`, mais celui de l'`OM`, et la valeur `2004` est la saison du `LOSC`.

Sortie structurée Le résultat d'une tâche d'extraction peut prendre la forme d'un arbre. Il s'agit d'une *sortie structurée*.

Considérons le document `XHTML` de la figure 1.9 qui contient une liste de livres Dans cet exemple, la tâche d'extraction consiste à obtenir pour chaque livre son titre et ses auteurs. Autrement dit, il s'agit d'extraire la liste des livres avec pour chaque livre la

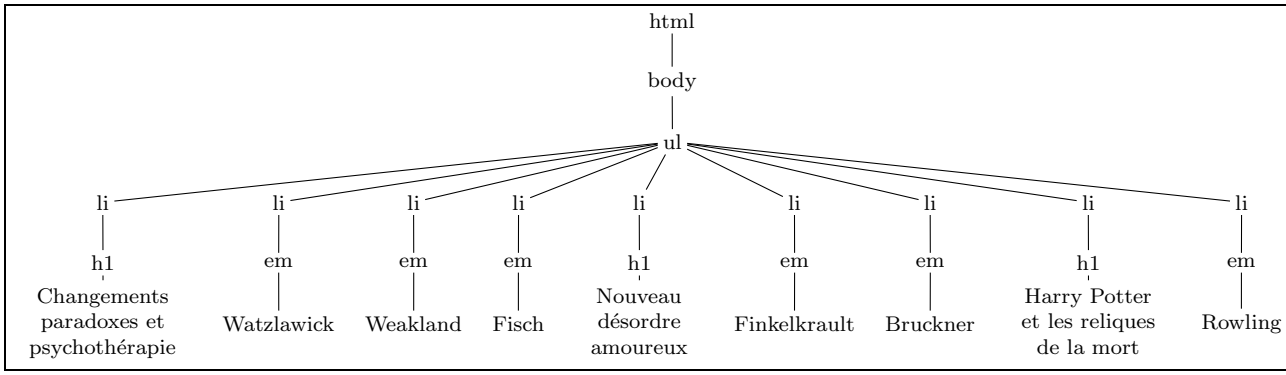


FIG. 1.9 – Un document `XHTML` contenant une liste de livres

liste de couples (titre,liste des auteurs). Étant donné que le nombre d'auteurs varie d'un livre à l'autre, il est malaisée de représenter l'information qu'on souhaite extraire par des n-uplets.

Dans ce cas une représentation sous la forme d'un arbre est plus adaptée, comme l'illustre la figure 1.10. Elle permet également d'apprécier la difficulté d'une tâche d'extraction d'information à sortie structurée. Certaines balises du document d'entrée ont disparu et ne sont pas présentent dans la sortie, comme `html` et `body`. Les autres balises ont été renommé, par exemple la balise `h1` du document `HTML` est renommée en une balise `titre` (du livre) dans le document `XML` de sortie. Ou encore la balise `li` qui devient `livre`, et la balise `em` se transforme en `auteur`. On constate également que les auteurs d'un même livre ont été regroupé sous une balise `auteurs`, qui ne correspond à aucun élément du document d'entrée.

Ainsi une tâche d'extraction d'information à sortie structurée est la transformation, potentiellement complexe, du document semi-structuré d'entrée en un arbre en sortie.

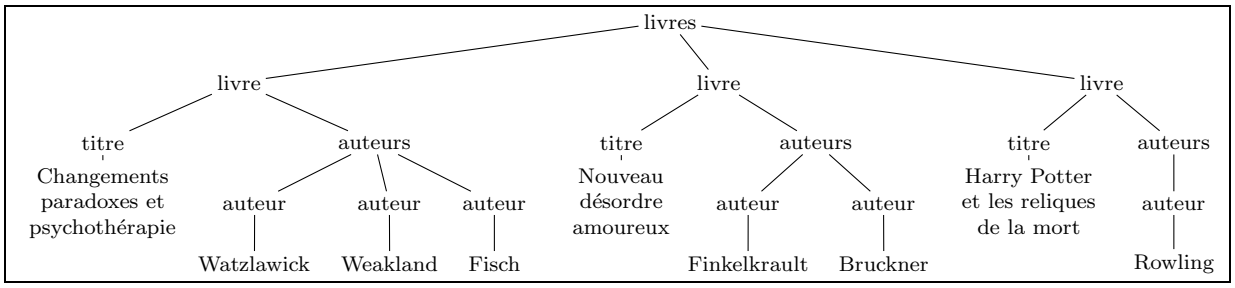


FIG. 1.10 – Sortie structurée extrait du document `XHTML` de la figure 1.9

1.3.3 Bilan

Pour représenter les documents semi-structurés, on utilisera dans nos approches la représentation arborescente qui exploite pleinement la structure des documents. Comme on l'a vu, la représentation à l'aide du rendu n'est pas stable et délicate à calculer. La représentation sous la forme d'une séquence ignore la structure arborescente des documents semi-structurés, ce qui limite son expressivité.

Parmi les différentes tâches d'extraction que nous avons illustré, dans cette thèse on s'intéresse à l'extraction n-aire. Les données n-aires sont fréquentes sur le Web. C'est par exemple le cas des données de la table du document de la figure 1.5 (page 15). Un tel document résulte de la publication sous forme de pages Web du contenu d'une base de données. Les documents semi-structurés produits par les suites bureautiques (comme OpenOffice) contiennent aussi des données n-aires, notamment les feuilles de calcul des tableurs. C'est encore le cas des rapports statistiques produits par les outils de reporting (comme JasperReport).

Les tâches n-aires sont plus difficiles que les tâches unaires car il faut à la fois extraire correctement chaque composante et aussi les ré-associer de manière appropriée pour former des n-uplets corrects. Comme on l'a illustré sur un exemple, certaines composantes peuvent ne pas avoir de valeur pour un n-uplet. Il s'agit des valeurs manquantes. Elles sont fréquentes dans les données réelles et participent à la difficulté de la tâche.

L'extraction n-aire est moins complexe que les tâches à sortie structurée. Cependant, dans cette thèse, nous considérons l'extraction d'information n-aire comme une transformation du document semi-structuré d'entrée en une table de n-uplets. Il s'agit là d'un cas particulier de transformation d'arbre (celui représentant le document semi-structuré d'entrée) en arbre, la table de tuples de sortie pouvant aisément se représenter sous la forme d'un arbre.

1.4 Conception d'extracteurs

Quatre approches permettent d'élaborer un extracteur avec un degré décroissant de l'intervention de l'utilisateur :

- l'écriture manuelle à l'aide, par exemple, d'un langage de manipulation de ou d'interrogation des documents ou encore d'un langage de programmation ;
- la spécification assistée qui guide l'utilisateur pour définir l'extracteur sans l'écrire manuellement ;
- l'induction supervisée qui produit par apprentissage un extracteur à partir d'un ensemble de documents annotés manuellement par l'utilisateur, *i.e.* des documents dans lesquels l'utilisateur a indiqué quelles sont les données à extraire ;
- l'induction non supervisée qui se passe de l'intervention de l'utilisateur.

1.4.1 Écriture manuelle

L'écriture manuelle d'extracteurs est possible à l'aide d'un langage de programmation (comme `Python`) et d'expressions régulières, qui permettent l'extraction de portions

de textes. Les langages d'interrogation et de transformation, comme `XSLT` et `XQUERY`, des dialectes `XML` sont également utilisés pour écrire des programmes d'extraction d'information.

Considérons la page Web de la figure 1.11 dont on souhaite extraire les triplets (`Club`,`Saison`,`Points`).

Club	Rennes	Rennes	Le Mans	Niort	Bastia	Martigues
League	1	1	2	2	1	2
Season	1999-2000	1999-2000	2002-2003	2001-2002	2000-2001	2000-2001
Round	17	6	27	36	13	7
Date	-	-	02/22/2003	Saturday, April 13th, 2002	-	-
Rank	5	10	3	15	4	13
Points	26	8	47	42	21	7
Wins	7	2	13	9	6	1
Draws	5	2	8	15	3	4
Losses	5	2	6	12	4	2
Goals Scored	25	8	33	36	15	8
Goals Conceded	22	9	26	37	12	9
Goal Difference	3	-1	7	-1	3	-1

FIG. 1.11 – Une partie d'une page Web de laquelle on souhaite extraire les triplets (`Club`,`Saison`,`Points`)

Comme on peut le constater, les données sont stockées dans une table et les six triplets à en extraire sont :

- (`Rennes`,`1999-2000`,`26`)
- (`Rennes`,`1999-2000`,`8`)
- (`Le Mans`,`2002-2003`,`47`)
- (`Niort`,`2001-2002`,`42`)
- (`Bastia`,`2000-2001`,`21`)
- (`Martigues`,`2000-2001`,`7`).

Ils sont stockés en colonne dans la table : chaque triplet est contenu dans une colonne de la table. La figure 1.12 présente le code source `HTML` de cette page Web,

La figure 1.13 présente une requête `XQUERY` permettant d'extraire les triplets désirés du document de la figure 1.12. Sans rentrer dans les détails techniques, précisons le fonctionnement de cette requête :

- tout d'abord le nombre n de triplets à extraire est évalué en comptant le nombre de `Club` (seconde ligne de la requête) ;
- ensuite les tuples sont extraits un à un à l'aide d'une boucle *pour* de 1 à n.

L'association des composantes en un triplet est déterminée par le fait qu'un club, une saison et un nombre de points doivent être combinés s'ils sont dans la même colonne. La sortie, *i.e.* la liste des n-uplets extraits, est produite sous la forme d'un document `XML`, présenté par la figure 1.14.

On constate ainsi qu'écrire une telle requête nécessite la connaissance et l'analyse de la structure du document ainsi que de l'organisation des données. Une telle ap-

```
<html> <body> <table>
    <tr> <th>Club</th> <td>Rennes</td> <td>Rennes</td> <td>Le Mans</td>
        <td>Niort</td> <td>Bastia</td> <td>Martigues</td> </tr>
    <tr> <th>League</th> <td>1</td> <td>1</td> <td>2</td>
        <td>2</td> <td>1</td> <td>2</td> </tr>
    <tr> <th>Season</th> <td>1999-2000</td> <td>1999-2000</td> <td>2002-2003</td>
        <td>2001-2002</td> <td>2000-2001</td> <td>2000-2001</td> </tr>
    <tr> <th>Round</th> <td>17</td> <td>6</td> <td>27</td>
        <td>36</td> <td>13</td> <td>7</td> </tr>
    <tr> <th>Date</th> <td>-</td> <td>-</td> <td>02/22/2003</td>
        <td>Saturday, April 13th, 2002</td> <td>-</td> <td>-</td> </tr>
    <tr> <th>Rank</th> <td>5</td> <td>10</td> <td>3</td>
        <td>15</td> <td>4</td> <td>13</td> </tr>
    <tr> <th>Points</th> <td>26</td> <td>8</td> <td>47</td>
        <td>42</td> <td>21</td> <td>7</td> </tr>
    <tr> <th>Wins</th> <td>7</td> <td>2</td> <td>13</td>
        <td>9</td> <td>6</td> <td>1</td> </tr>
    <tr> <th>Draws</th> <td>5</td> <td>2</td> <td>8</td>
        <td>15</td> <td>3</td> <td>4</td> </tr>
    <tr> <th>Losses</th> <td>5</td> <td>2</td> <td>6</td>
        <td>12</td> <td>4</td> <td>2</td> </tr>
    <tr> <th>Goals Scored</th> <td>25</td> <td>8</td> <td>33</td>
        <td>36</td> <td>15</td> <td>8</td> </tr>
    <tr> <th>Goals Conceded</th> <td>22</td> <td>9</td> <td>26</td>
        <td>37</td> <td>12</td> <td>9</td> </tr>
    <tr> <th>Goal Difference</th> <td>3</td> <td  >-1</td> <td>7</td>
        <td>-1</td> <td>3</td> <td>-1</td> </tr>
</table> </body> </html>
```

FIG. 1.12 – Code source HTML de la page Web de la figure 1.11

```
<tuples>
    let $nbcol:=count(/table/tbody/tr[th="Club"]/td)
    for $i in (1 to \$nbcol)
    return
    <tuple>
        {
            <club>{/table/tbody/tr[th="Club"]//td[position()=$i]/text()}</club>,
            <saison>{/table/tbody/tr[th="Season"]/td[position()=$i]/text()}</saison>
            <points>{/table/tbody/tr[th="Date"]/td[position()=$i]/text()}</points>
        }
    </tuple>
</tuples>
```

FIG. 1.13 – Requête `XQUERY` permettant d'extraire les triplets (`Club`,`Saison`,`Date`) du document `HTML` de la figure 1.12

proche n'est pas du tout générique mais au contraire très spécifique à la fois à la tâche d'extraction d'information considérée et aux documents traités.

Au delà de l'aspect technologique, l'écriture manuelle d'extracteurs nécessite une analyse des documents à traiter et de leur organisation, afin de résoudre le problème d'extraction d'information. C'est une tâche difficile, source d'erreurs, qui nécessite des connaissances et une expertise qui n'est pas à la portée de tout un chacun. Enfin les changements dans la structure des documents du Web sont fréquents et rendent délicate la maintenance d'extracteurs conçus manuellement.

1.4.2 Spécification assistée

Les approches par spécification assistée, comme XWrap [59], W4F [74] et Lixto [4, 38], guident l'utilisateur au cours du processus de conception de l'extracteur. Par le biais d'une interface graphique, l'outil de spécification dialogue avec l'utilisateur qui lui désigne les éléments à extraire et lui livre également des connaissances sur les documents à traiter et leur organisation. Ces systèmes proposent souvent des motifs d'extraction, à base d'expressions régulières sur les mots, déjà écrits et l'utilisateur choisit le motif adapté selon le type de la donnée à extraire. On trouve par exemple des motifs pour l'extraction de valeurs numériques, de prix, d'URL, les numéros de téléphone et de fax, *etc* De cette interaction, résulte la production d'un extracteur pour la tâche considérée, sans que l'utilisateur n'ait ni à manipuler directement le formalisme sous-jacent ni même à le connaître.

Tout comme l'écriture manuelle d'extracteurs, la spécification assistée impose, dans une moindre mesure, à l'utilisateur de réaliser une grande partie de l'expertise de la tâche d'extraction et de l'analyse des documents à traiter. De plus ce genre d'outils suppose une capacité d'abstraction et une démarche algorithmique qui le réserve encore à des experts.

```
<tuples>
    <tuple>
        <club>Rennes</club>
        <saison>1999-2000</saison>
        <points>26</points>
    </tuple>
    <tuple>
        <club>Rennes</club>
        <saison>1999-2000</saison>
        <points>8</points>
    </tuple>
    <tuple>
        <club>Le Mans</club>
        <saison>2002-2003</saison>
        <points>47</points>
    </tuple>
    <tuple>
        <club>Niort</club>
        <saison>2001-2002</saison>
        <points>42</points>
    </tuple>
    <tuple>
        <club>Bastia</club>
        <saison>2000-2001</saison>
        <points>21</points>
    </tuple>
    <tuple>
        <club>Martigues</club>
        <saison>2000-2001</saison>
        <points>7</points>
    </tuple>
</tuples>
```

FIG. 1.14 – Résultat produit par l'évaluation de la requête `XQUERY` de la figure 1.13 sur le document `HTML` de la figure 1.12

1.4.3 Induction supervisée d'extracteurs

L'induction d'extracteurs [52] décharge l'utilisateur de toute expertise. L'extracteur est produit automatiquement par un algorithme d'apprentissage à partir des documents annotés par l'utilisateur. Les algorithmes d'induction exploitent les régularités syntaxiques et/ou structurelles des documents afin de repérer les données à extraire.

Une approche syntaxique pour l'extraction est possible en raison de la structure des documents `HTML` et `XML`. Cette structure est bien souvent d'une très grande régularité. C'est par exemple le cas des pages `HTML` produites automatiquement selon un patron de mise en forme comme dans les outils de publication collaborative ou les systèmes de gestion de contenu. Les documents du Web caché possèdent également une très grande régularité. Le Web caché est l'ensemble des pages Web accessibles uniquement par des points d'entrée spécifiques qui sont des interfaces de recherche, comme par exemple des formulaires `HTML`. Les pages de résultats issus des moteurs de recherche ou des formulaires d'interrogation présents sur de nombreux sites, comme par exemple les sites de commerce, les librairies numériques, ou encore les sites de statistiques appartiennent du Web caché. Les résultats renvoyés par le traitement de ces formulaires proviennent la plupart du temps d'une base de données et sont présentés dans des pages Web produites dynamiquement par programme. Encore une fois, le patron de mise en page des données assure une grande régularité dans la structure des pages que l'on peut exploiter pour l'extraction d'information.

Parmi les techniques d'apprentissage automatique utilisées pour l'induction supervisée d'extracteurs, on peut citer l'inférence grammaticale sur les chaînes [16] et sur les arbres [48, 12, 14, 56], les modèles probabilistes d'annotations de séquences [30, 17, 68, 75] et de structures arborescentes [45], la programmation logique inductive [11, 43, 81] et la classification supervisée [29, 27, 60, 61, 35, 36]. D'autres méthodes ont développé des algorithmes spécifiques pour l'apprentissage d'extracteurs. On trouve notamment des approches sur les chaînes à base de délimiteurs [51, 42, 78, 52, 65], des approches à base d'expressions régulières [40] et l'apprentissage de motifs arborescents [39].

La critique fréquemment adressée aux algorithmes d'induction supervisée d'extracteurs est l'importance de l'effort d'annotation de la part de l'utilisateur. Même si de nombreux systèmes [51, 11, 29, 48, 81, 27, 61, 56] nécessitent des documents complètement annotés, *i.e.* toutes les données à extraire sont marquées, pour l'apprentissage de l'extracteur, il existe également des systèmes fonctionnant à partir de quelques annotations seulement [57, 43, 12, 36], dans le but de minimiser le nombre d'annotations.

1.4.4 Induction non supervisée

L'induction d'extracteur peut être complètement automatisée et se passer de l'intervention de l'utilisateur. Il s'agit de l'*induction non supervisée* qui produit un extracteur à partir des documents sur lesquels porte l'extraction.

Parmi les systèmes existants, on peut citer ROADRUNNER [22] et MDR [88]. Les extracteurs induits par ces systèmes produisent en sortie des données structurées sous la forme d'une table, avec éventuellement des imbrications. Ces algorithmes d'induc-

tion déterminent quelles sont les données à extraire en analysant la structure et les régularités des documents. Ils sont souvent basés sur des méthodes d'alignement (de chaînes ou d'arbres) des documents [3, 15, 57, 15, 88] ou bien sur des techniques d'inférence grammaticale [22].

Le principal avantage des algorithmes d'induction non supervisée est leur automatisation complète. Comme l'utilisateur n'est pas nécessaire à leur fonctionnement, ils peuvent être intégrés à des chaînes de traitements automatiques des documents du Web. Il peut s'agir, par exemple, de l'extraction des données du Web caché, de l'alimentation d'un entrepôt de données, *etc.*

Comme le souligne [87], les extracteurs obtenus de manière non supervisée sont moins précis que ceux issus d'un processus d'induction supervisée. Généralement, les extracteurs induits par une méthode non supervisée se contentent d'extraire la zone du documents contenant les données et au mieux de la partitionner en sous-zones. Un post-traitement est souvent nécessaire pour identifier parmi les données extraites, celles qui sont réellement la cible de l'extraction.

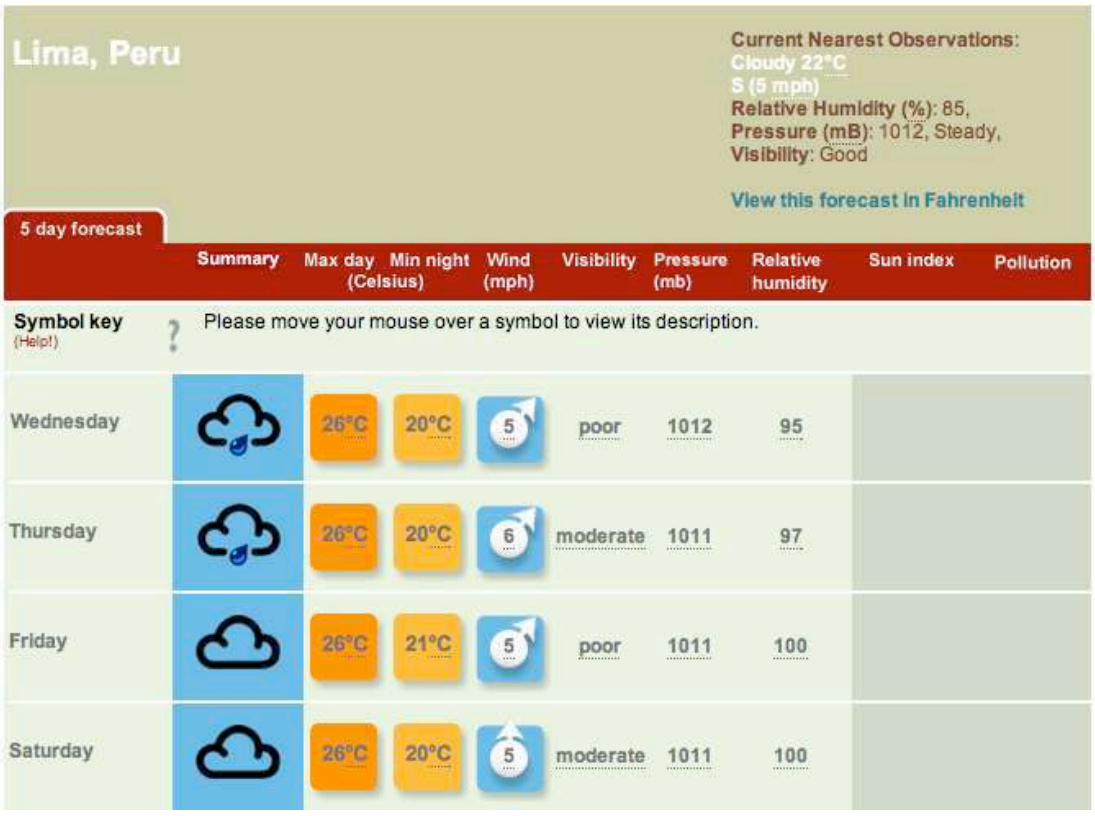


FIG. 1.15 – Une page du site Web météorologique de la BBC

Par exemple, la figure 1.16 montre la table des données extraites par ROADRUNNER

à partir d'une page du site météorologique de la BBC[2] présentée par la figure 1.15.

La première colonne de la table de la figure 1.16 indique le nom des champs (choisis automatiquement par ROADRUNNER) et la seconde les valeurs extraites correspondantes. On s'aperçoit que les données météorologiques contenu dans la table de la page de la figure 1.15 sont extraites par ROADRUNNER sous la forme d'une liste. La structure tabulaires des données est ainsi perdue. De plus si l'on s'intéressait à l'extraction des couples `(Jour,Température Maximale)`, qui correspondent respectivement aux première et troisième colonnes de la table de la page de la figure 1.15, on s'aperçoit que les valeurs des jours n'ont pas été extraites et qu'un post-traitement est nécessaire pour déterminer parmi les valeurs extraites celles qui correspondent à la température maximale et aussi pour re-constituer les couples.

_E_	Lima, Peru
_F_	cloudy
_G_	22
_H_	S (5
_I_	: 85,
_J_	: 1012, Steady,
_K_	: Good
_O_	26
_Q_	20
_S_	5
_T_	poor
_U_	1012
_V_	95
_W_	
_X_	
_Y_	26
_Z_	
_A1_	20
_B1_	
_C1_	6
_D1_	1011

FIG. 1.16 – Résultats de l'extraction de la page Web de la figure 1.15 par ROADRUNNER. Les valeurs extraites sont dans la seconde colonne.

On pourrait argumenter qu'une méthode non supervisée peut servir de pré-traitement à une méthode supervisée, en identifiant les régions contenant les données, afin de

[2] http://www.bbc.co.uk/weather

réduire la taille des documents à traiter. Cependant extraire les informations d'après le résultat d'un algorithme non supervisé peut-être aussi difficile qu'extraire les mêmes données dans le document d'origine. La page de résultats de la figure 1.16 l'illustre bien : la structure des données extraites est beaucoup moins riche que celle du document de la figure 1.15.

1.4.5 Bilan

Parmi les quatre méthodes de conception d'extracteurs présentées, l'écriture manuelle et la spécification assistée sont écartées car elles ne s'adressent pas à un utilisateur néophyte. L'induction non supervisée n'est pas retenue car les extracteurs induits manquent de précision et un post-traitement peut être nécessaire pour filtrer ou mieux structurer les données extraites. Cette thèse se focalise donc sur l'induction supervisée d'extracteurs n-aires pour les documents semi-structurés.

Cette approche nécessite la connaissance d'exemples de données à extraire et donc une annotation des documents d'apprentissage par l'utilisateur. Afin que la tâche d'annotation ne soit pas laborieuse, il faut que l'apprentissage puisse se faire à partir de peu d'exemples. Récemment différents systèmes d'induction supervisée d'extracteurs ont atteints avec succès cet objectif à l'aide d'une approche interactive entre l'utilisateur et l'algorithme d'apprentissage [54, 12]. Cette thèse s'inscrit dans cette démarche. Elle fixe comme objectif la proposition d'un algorithme d'induction supervisée d'extracteurs n-aires, pour les documents semi-structurés, nécessitant peu d'annotations.

1.5 Approche supervisée pour l'induction d'extracteurs

Nous avons choisi de réaliser l'induction d'extracteurs à l'aide de technique d'apprentissage supervisé, avec la contrainte que l'utilisateur annote le moins d'exemples possibles. On distingue deux types d'exemples pour l'induction d'extracteurs :

- les exemples positifs qui sont les données à extraire ;
- les exemples négatifs (ou contre-exemples) qui correspondent aux données que l'utilisateur ne souhaite pas extraire.

Une contrainte supplémentaire est celle de pouvoir utiliser n'importe quel algorithme d'apprentissage supervisé existant en tant que brique d'apprentissage élémentaire et comme une boite noire. L'objectif est ici de bénéficier de l'existant en la matière d'algorithmes d'apprentissage et de son évolution future.

Apprentissage par positifs seuls Si l'on considère qu'il est plus naturel à l'utilisateur final d'annoter des exemples positifs dans les documents d'apprentissage, que d'indiquer des exemples négatifs, alors les approches d'apprentissage par exemples positifs seuls semblent adaptées pour traiter le problème [24, 58]. Cependant ces techniques d'apprentissage sont fondées sur une estimation de la densité des exemples négatifs parmi les exemples non étiquetés, ce qui nécessite une modification profonde des al-

gorithmes d'apprentissage et qui va à l'encontre de notre choix de d'utiliser n'importe quel algorithme d'apprentissage supervisé tel qu'il est.

De plus les systèmes d'induction n-aire par positifs seuls, comme [51, 42, 39], manipulent implicitement des exemples négatifs et font des hypothèses fortes sur les documents d'entrée et la disposition des tuples. Ce dernier point est particulièrement important car ces hypothèses limitent l'expressivité des extracteurs que l'on peut obtenir.

C'est pourquoi une telle technique d'apprentissage n'a pas été choisie dans cette thèse, et qu'on a favorisé une approche à base d'exemples positifs et négatifs. Examinons maintenant un cadre d'apprentissage qui justifie l'utilisation d'exemples négatifs.

Apprentissage interactif Un argument supplémentaire pour motiver le recours à un algorithme d'apprentissage utilisant des exemples et des contre-exemples apparaît lorsqu'on place l'utilisateur dans un cadre interactif, comme celui de [12].

Dans un tel cadre, l'induction de l'extracteur, ou extracteur hypothèse, est amorcée à partir de l'annotation initiale de quelques exemples positifs. Il est fort probable qu'avec seulement quelques exemples de données à extraire, l'algorithme d'apprentissage généralise mal et induise un extracteur incorrect. Afin de vérifier la validité de l'extracteur hypothèse, ce dernier est appliqué aux documents d'apprentissage et les données extraites sont soumises à l'expertise de l'utilisateur. Deux types d'erreur peuvent survenir :

- des données à extraire ne l'ont pas été : dans ce cas l'utilisateur annote quelques unes d'entre elles (annotation d'exemples positifs) ;
- des données qui ne sont pas extraite l'on été : dans ce cas l'utilisateur signale à l'algorithme d'apprentissage que ces données ne l'intéressent pas en annotant certaines comme exemple négatif.

Après avoir corrigé quelques unes des erreurs de l'hypothèse courante, l'utilisateur relance l'apprentissage. Ce processus est itéré jusqu'à l'obtention d'un extracteur hypothèse correct.

Exemples négatifs et induction d'extracteurs n-aires Dans le cadre interactif que nous venons de présenter, le rôle des exemples négatifs apparaît clairement, et leur emploi est naturel pour l'utilisateur. Cependant un problème majeur advient. Pour l'induction d'extracteurs n-aires, le nombre d'exemples négatifs est exponentiel l'arité des n-uplets à extraire, notée n. Devant ce fait, on ne peut que constater l'impossibilité d'employer une approche directe de l'apprentissage d'extracteurs n-aires, avec comme exemples des n-uplets. En effet, la très forte disproportion entre le nombre d'exemples positifs et négatifs est telle que choisir comme critère de discrimination des exemples la règle *tout exemple est négatif* ne conduit à commettre qu'une très faible erreur. En formulant cette règle autrement, cela revient à dire qu'il n'y a pas de données à extraire, ce qui n'est guère pertinent pour l'extraction d'information.

Approche itérative Pour contourner ce problème d'explosion du nombre de négatifs, une approche envisageable est de décomposer l'induction en plusieurs apprentissages dont le nombre d'exemples négatifs n'est pas exponentiel.

Cette décomposition est possible en réalisant l'extraction n-aire de manière incrémentale selon une boucle croissante sur la taille des n-uplets. Le processus est amorcé par l'extraction des singletons, à partir desquels seront extraits des couples qui eux-mêmes serviront à l'extraction de triplets, *etc.*

L'algorithme d'apprentissage associé à ce procédé d'extraction est également fondé sur une boucle croissante sur la taille des tuples, de 1 à n. Le problème d'apprentissage est décomposé en n problèmes d'apprentissage. Pour le problème i, les exemples positifs sont les tuples à extraire restreint aux i-ères composantes. Les exemples négatifs sont obtenus par un procédé de construction qui consiste à étendre les tuples positifs de taille $i-1$ en des tuples de taille i qui ne sont pas des exemples positifs. Ainsi, le nombre d'exemples négatifs est contrôlé et borné à chaque étape.

Autres approches D'autres approches que la notre sont également basées sur un processus de décomposition. L'apprentissage d'un extracteur n-aire est décomposé en l'apprentissage de n extracteurs unaires, un pour chaque composante [65, 19]. Lors de l'extraction, les extracteurs fonctionnent en parralèle. Les valeurs extraites pour chaque composante sont ensuite combinées pour former des n-uplets. Ce processus de recombinaison peut être spécifié par l'utilisateur [65, 19] et/ou basé sur des hypothèses sur l'agencement des n-uplets dans les documents [65]. Ainsi la re-composition des tuples n'est pas assurée par un processus d'apprentissage mais par un post-traitement. Nous pensons qu'il est préférable d'apprendre simultanément à extraire les valeurs de chaque composante et à constituer les tuples. C'est pourquoi nous préférons dans cette thèse une approche sans post-traitement, et que nous écartons une telle méthode de décomposition.

Parmi les nombreuses autres approches utilisées, nous pouvons citer celles par annotation, soit à l'aide d'automates d'abres [13], soit à l'aide de modèles probabilistes [30, 17, 75, 50, 46]. L'esprit des approches par annotation peut se résumer par la devise suivante : annoter pour extraire. Dans le cas le plus simple, les différentes parties du document sont annotées par une étiquette binaire dont les deux valeurs sont "à extraire" et "à ne pas extraire". Cependant les limites de l'expressivité de ces approches apparaissent rapidement.

Les automates d'arbres permettent de poser des requêtes sur les documents semi-structurés [12]. Mais ils sont limités aux requètes régulières. Or certaines tâches d'extraction d'information n-aire ne le sont pas et ne peuvent se résoudre qu'en comptant des positions dans un arbre (comme on l'a illustré sur la figure 1.5 à la page 16). Les modèles probabilistes d'annotations souffrent des mêmes maux. Bien que des travaux comme [46], basés sur un modèle d'annotation probabiliste pour les arbres[3], permettent de faire des transformations complexes d'arbres, ils ne peuvent pas non plus traité les tâches d'extraction n-aire non régulières. Les hypothèses sur le document supposent par exemple que les valeurs des tuples ne s'entremêlent pas dans le feuillage de l'arbre. Or cette hypothèse est forte et ne permet pas de répondre correctement à l'extraction de données dans une table croisée, comme par exemple une table de distance ville à ville ou encore la table de la figure 1.5, dans laquelle les valeurs des tuples stockés dans la table sont enchevêtrées. C'est pourquoi les approches par annotation ont été écarté.

[3] TreeCRF : http://treecrf.gforge.inria.fr/

1.6 Objectifs et contribution

La thèse défendue dans ce mémoire est qu'il est possible de concevoir des algorithmes d'apprentissage de programmes d'extraction n-aire pour les documents semi-structurés, qui est une classe non triviale de transformation d'arbres, de manière supervisée et avec peu d'intervention de l'utilisateur.

Les documents semi-structurés ont une structure arborescente. Hors peu de systèmes d'induction supervisée d'extracteurs en tirent partie. La plupart d'entre eux considèrent les documents comme une séquence mélangeant balises et contenu [51, 42, 40, 78, 65]. Plus récemment sont apparus des algorithmes d'induction exploitant pleinement la structure d'arbre des documents semi-structurés [43, 48, 81, 12, 39, 56, 36]. Cette thèse s'inscrit dans ce courant et soutient l'idée que l'exploitation de la structure des documents semi-structurés permet d'induire des extracteurs expressifs et performants.

L'induction est réalisée à l'aide d'algorithmes d'apprentissage automatique de classification supervisée. Ce choix est motivé à la fois par le succès des approches d'extractions fondée sur la classification, mais surtout par la volonté d'utiliser des algorithmes d'apprentissage existants et connus. Bien que le codage de exemples d'apprentissage en attribut-valeur prenne en compte la nature arborescente des documents semi-structurés, il est générique et intègre peu de connaissance de base. Cependant toute nouvelle connaissance est facilement intégrable. Notre représentation des données est adaptative.

Dans notre approche, l'extraction n-aire est réalisée de manière incrémentale au cours d'une boucle croissante sur la taille des n-uplets. Ce procédé d'extraction ne fait aucune hypothèse sur la disposition des données dans les documents. Aucun post-traitement n'est effectué : notre algorithme réalise en même temps l'extraction des composantes et leur combinaison en n-uplets.

Précisons qu'un extracteur obtenu par PAF, notre système, est utilisable tel quel, comme une boite noire, avec en entrée des documents `HTML` ou `XML`, et en sortie l'ensemble des n-uplets extraits. De plus le système PAF est implémenté dans un cadre interactif qui permet l'induction à partir d'un faible nombre d'interactions. L'utilisateur fournit quelques annotations qui servent d'amorce à l'apprentissage d'un extracteur hypothèse. Ici commence une boucle d'interaction dans laquelle l'utilisateur corrige les erreurs de l'hypothèse courante et relance l'apprentissage jusqu'à l'obtention d'une hypothèse correcte. PAF permet d'apprendre des extracteurs n-aires performants à partir de peu d'exemples.

Les résultats expérimentaux montrent que PAF atteint les performances des meilleurs systèmes n-aires. De plus son procédé d'extraction reste applicable et efficace même lorsque l'organisation des données dans les documents semi-structurés est complexe. L'évaluation expérimentale montre également que le cadre interactif de PAF permet de réduire l'effort d'annotation de l'utilisateur, tout en préservant la qualité des extracteurs induits.

1.7 Plan de la thèse

Cette thèse commence par le chapitre 2. Il illustre le principe sur lequel est fondé notre approche de l'extraction n-aire et que l'on peut résumer par la maxime : classer pour extraire. Il présente à la fois les techniques de classification supervisée utilisées dans cette thèse et les systèmes d'induction d'extracteurs basés sur la classification. Dans le chapitre 2, et uniquement celui-ci, nous nous limiterons au cas de l'extraction unaire dans les documents non-structurés. De l'étude des systèmes existants pour cette tâche spécifique, nous dégagerons une approche générique pour formuler un problème d'extraction comme un problème de classification.

Le chapitre 3 propose la première contribution de cette thèse qui est l'étude de l'agencement des tuples dans un document structuré de manière arborescente. Nous avons identifié cinq cas d'organisation de base pour stocker des n-uplets de valeurs dans un arbre, que nous confrontons à quelques systèmes d'extraction n-aire, présentés plus en détail dans le chapitre A. Ainsi nous montrons qu'ils sont tous spécialisés sur certaines organisations des tuples. Aucun d'entre eux n'est capable de traiter l'intégralité des cinq cas élémentaires de stockage des données.

Dans le chapitre 4 nous présentons notre approche de l'induction d'extracteurs n-aires. Deux algorithmes sont proposés : celui d'extraction incrémentale et l'algorithme d'apprentissage correspondant, qui à partir d'un ensemble de documents complètement annotés. Ce dernier fonctionne à partir d'un ensemble de documents complètement annotés. Ils font l'objet d'une implémentation au sein du prototype PAF. La validité de notre approche est montrée par une étude expérimentale.

Enfin cette thèse se termine par le chapitre 5 en montrant comment modifier l'algorithme d'apprentissage du chapitre 4 pour induire un extracteur à partir de documents partiellement annotés. En effet dans la pratique on souhaite obtenir un extracteur à partir du plus petit nombre d'exemples possible. L'amélioration proposée conduit à un système interactif avec l'utilisateur. Nous montrons expérimentalement que cette version interactive de PAF est aussi performante que celle du chapitre 4 mais nécessite moins d'annotations.

Chapitre 2

Induction d'extracteurs et classification supervisée

2.1 Introduction

Plusieurs techniques d'apprentissage automatique ont été appliquées avec succès à l'induction supervisée d'extracteurs. Parmi elles, nous nous focalisons sur la classification supervisée que nous avons choisi comme brique d'apprentissage de base dans nos algorithmes d'induction. L'apprentissage de classificateurs est un domaine de recherche très étudié. De nombreux algorithmes d'apprentissage sont disponibles et leurs propriétés théoriques et/ou empiriques sont connues. Bénéficier de ces connaissances et réutiliser des algorithmes existants et performants nous semble pertinent.

Le but de ce chapitre est double. D'une part, nous introduisons les notions de classification supervisée utilisées par notre approche. D'autre part, nous étudions comment la formulation du problème d'extraction d'information en un problème de classification supervisée est réalisée par plusieurs systèmes d'induction d'extracteurs. Cette analyse de l'existant met en évidence une architecture commune et dégage les principes que nous utiliserons pour concevoir les algorithmes d'extraction et d'induction du chapitre 4.

2.2 Classification Supervisée

Cette section n'apporte pas de contribution personnelle. Elle présente succinctement la classification supervisée, le langage attribut valeur de description des exemples et les algorithmes d'apprentissage que nous utiliserons comme brique d'apprentissage de base dans nos algorithmes d'induction d'extracteurs.

2.2.1 Généralités

Classer un objet consiste à l'affecter au groupe d'objets auquel il appartient. Cela revient à lui associer une classe. L'apprentissage de cette association objet/classe peut

se faire à partir d'objets dont la classe est déjà connue. On se trouve alors dans le cadre de la *classification supervisée* [62, 21].

Le modèle d'apprentissage de la classification supervisée suppose l'existence d'un *concept cible* f que l'on cherche à découvrir à partir d'un ensemble d'objets ou *exemples* déjà classés. On peut voir f comme une fonction de l'ensemble des exemples vers l'ensemble des classes [1]. L'apprentissage consiste alors à approximer au mieux f par une fonction hypothèse h. Par la suite la fonction h sera utilisée pour prédire la classe d'objets dont la classe est inconnue.

D'après [18], la définition d'un problème d'apprentissage passe par le choix de deux langages :

– celui permettant de représenter les exemples ;
– le langage de représentation des hypothèses.

À chaque exemple, on associe sa description $x \in \mathcal{X}$ et sa classe $y \in \mathcal{Y}$. L'ensemble $\mathcal{X}$ est *l'espace de description des exemples* et $\mathcal{Y}$ est *l'ensemble des classes possibles*. Le couple (x, y) est un *exemple étiqueté*, et l'ensemble de ces couples, noté S, est *l'ensemble d'apprentissage*. Dans le cas de la classification binaire, il n'y a que deux classes possibles. Généralement on note $\mathcal{Y} = \{-1, +1\}$. On dit alors que x est un *exemple positif* si $y = +1$, et un *exemple négatif*, ou contre exemple, si $y = -1$.

Classer consiste à associer à tout élément de $\mathcal{X}$ un élément de $\mathcal{Y}$. Ainsi la fonction cible f est définie sur l'ensemble $\mathcal{X}$ et prend ses valeurs dans $\mathcal{Y}$, *i.e.* $f : \mathcal{X} \mapsto \mathcal{Y}$. Pour un exemple étiqueté (x, y) on a donc $y = f(x)$. Le but de l'apprentissage en classification supervisée est alors d'induire d'après un ensemble d'exemples étiquetés S une hypothèse $h : \mathcal{X} \mapsto \mathcal{Y}$ qui approche le plus possible la fonction cible f (dont on ne dispose pas). L'écart entre la h et f est évalué par une fonction d'erreur l. L'idéal serait d'évaluer h sur tous les exemples de $\mathcal{X}$. En pratique cette évaluation s'effectue sur les seuls exemples à disposition : ceux de l'ensemble d'apprentissage S. La fonction d'erreur l utilisée mesure alors *l'erreur empirique* de h par rapport à f sur l'ensemble d'apprentissage S, qui est le nombre de fois où h et f diffèrent sur la prédiction de la classe d'un exemple. Plus formellement, $l(h, f)$ est le cardinal de l'ensemble $\{x \in S \mid h(x) \neq f(x)\}$.

On parle également d'apprentissage de concept, car il s'agit de dégager un principe général à partir d'exemples et de contre-exemples particuliers. Le rôle de l'apprenant est donc de généraliser à partir des exemples d'apprentissage.

L'ensemble des hypothèses est noté H. C'est l'espace de recherche de l'algorithme d'apprentissage. Il est défini implicitement par la manière dont les hypothèses sont représentées. Généralement, le processus de généralisation opéré par l'apprenant est vu comme une recherche dans l'espace des hypothèses de celle qui correspond au mieux aux exemples d'apprentissage [63]. Il existe de nombreuses stratégies pour parcourir cet espace ou le réduire par un élagage. C'est de cette stratégie de recherche que dépend le succès d'un algorithme d'apprentissage. Son objectif est souvent de trouver un compromis entre l'hypothèse la plus générale et l'hypothèse la plus spécifique. L'apprenant doit suffisamment généraliser pour se détacher des données d'apprentissage, et ainsi éviter le *sur-apprentissage* qui est un apprentissage par cœur des données. Mais il doit aussi ne pas sur-généraliser pour ne pas trop s'éloigner du concept cible en apprenant une hypothèse trop générale. La littérature dans le domaine de la classification supervisée

[1] on se limite uniquement au cas où chaque exemple est associée à exactement une classe

propose une foison d'algorithmes d'apprentissage [62, 21].

Dans le cadre de cette thèse, nous retiendrons qu'à partir d'un ensemble d'exemples étiquetés en entrée, un apprenant induit en sortie un *classificateur*, qui est la fonction hypothèse retenue par le processus d'apprentissage. Ce classificateur permet de prédire la classe d'exemples non étiquetés. Il prend en entrée un exemple, et fournit en sortie la classe qu'il lui a associée. Une propriété importante que nous définissons maintenant est la *consistance* d'un classificateur.

Consistance d'un classificateur Un classificateur est dit *consistant* avec son ensemble d'apprentissage s'il ne commet aucune erreur de prédiction sur cet ensemble (voir [62] page 29). Autrement dit, un classificateur est consistant s'il classe correctement tous les exemples d'apprentissage. Plus formellement, soit c est le classificateur appris à partir de l'ensemble d'apprentissage $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$. Alors c est consistant si et seulement si pour tout $(x_i, y_i) \in S$ on a $c(x_i) = y_i$.

2.2.2 Représentation en attribut valeur

Nous présentons ici le langage attribut valeur (ou d'ordre zéro) qui est un langage de description des exemples.

Dans ce langage, un exemple est représenté par un *vecteur d'attributs*. Un attribut est toute quantité, valeur, ou mesure capable de décrire une caractéristique d'un exemple. Si l'exemple à représenter est un pingouin, alors la couleur de son bec, celle de son pelage, sa taille et son poids sont des attributs permettant de le décrire. La classe d'un exemple est généralement considérée aussi comme un attribut.

On distingue généralement deux types d'attributs :
- ceux à *valeur continue* ;
- les attributs à *valeur discrète*.

Un attribut continu prend ses valeurs dans R, alors qu'un attribut discret, ou catégoriel, prend ses valeurs parmi un ensemble de valeurs discrètes.

Un ensemble d'apprentissage peut ainsi être codé par une matrice. Chaque colonne correspond à un attribut, et une ligne est la représentation d'un exemple. La valeur à la croisée d'une ligne i et d'une colonne j est la valeur de l'attribut j pour l'exemple i. Il se peut que certains exemples ne possèdent pas de valeur pour un ou plusieurs attributs. Dans ce cas, une valeur spéciale, dite indéterminée et généralement notée par un point d'interrogation, est utilisée pour indiquer l'absence de valeur. Lorsque la valeur indéterminée apparaît dans une hypothèse, elle couvre toute valeur que peut prendre un exemple sur l'attribut concerné.

2.2.3 Quelques algorithmes d'apprentissage

Dans cette partie du chapitre, nous introduisons les algorithmes de classification supervisée que nous utilisons par la suite. Il s'agit uniquement d'algorithmes à base d'une représentation en attribut valeur des exemples. Pour plus de détails sur les nombreuses méthodes d'apprentissage proposées par la classification supervisée, nous renvoyons le lecteur à [62] ainsi qu'à [21].

2.2.3.1 Arbres de décision

Les arbres de décision sont des classificateurs très populaires, notamment en raison de leur grande lisibilité, qui rend leurs prédictions interprétables par un expert humain. Ils sont également résistants au bruit, qui peut porter à la fois sur les valeurs prises par les exemples pour certains attributs (certaines valeurs sont erronées), que sur la classe des exemples (certaines classes ne sont pas correctes).

Une hypothèse prend donc la forme d'un arbre dont chaque nœud interne spécifie un test sur un attribut. Chaque branche sortante du nœud est étiquetée par l'une des valeurs possibles prises par cet attribut, et conduit aux fils du nœud courant. Chaque feuille de l'arbre est étiquetée par une classe du problème de classification (une même classe peut apparaître dans des feuilles différentes). Chaque chemin, de la racine à une feuille, correspond donc à une conjonction de tests sur les attributs. L'arbre, quand à lui, est une disjonction des conjonctions de l'ensemble de ses chemins.

La classification d'un exemple consiste à suivre un chemin dans l'arbre, en partant de la racine jusqu'à une feuille. La valeur de l'attribut testé au nœud courant indique quelle branche sortante suivre. En se laissant ainsi guider, l'exemple descend dans l'arbre jusqu'à une feuille. La classe associée à cette dernière est la classe prédite pour l'exemple.

Il existe plusieurs algorithmes d'apprentissage pour les arbres de décisions [72, 71, 47, 73]. Cependant ils sont tous des variantes de l'algorithme générique 1, que l'on note A. Le principe de cet algorithme est de construire récursivement l'arbre de décision à partir de l'ensemble d'apprentissage S en choisissant pour chaque nœud un test qui partitionne les exemples.

L'algorithme A commence par la création d'un nouveau nœud n auquel sont associés tous les exemples de S (ligne 1). Ensuite, le critère d'arrêt de la construction de l'arbre enraciné en n est testé (ligne 2) grâce à la fonction $Arret$ qui prend l'ensemble S comme argument. Si le critère d'arrêt est vrai, alors le nœud n est une feuille, et il faut lui associer une classe, notée $n.classe$, parmi celles présentes dans S. Ce choix est réalisé par la fonction $SelectionClasse$ à la ligne 3. Si le critère d'arrêt est faux, alors il faut sélectionner le test du nœud n, noté $n.test$. Pour ce faire, tous les tests que l'on peut construire à partir des exemples associés à n sont évalués. Un test t est une fonction de l'ensemble des exemples S vers l'ensemble V des valeurs prises, notées $v_1, \ldots, v_k$, par le test, et l'on note $t(x) = v_i$ si $v_i \in V$ est la valeur prise par le test t pour l'exemple x.

Le test $n.test$ qui maximise le critère de sélection $SelectionTest$, généralement de nature statistique, est affecté au nœud n (ligne 5). La fonction $SelectionTest$ intègre un mécanisme de contruction des tests qui est examiné au paragraphe suivant. On considère l'ensemble des valeurs possibles de $n.test$ qui déterminent la partition $S_1, \ldots, S_k$ de l'ensemble des exemples S, selon valeur prise par $n.test(x)$ pour chaque exemple $x \in S$. À chaque partie S_i correspondra un nœud n_i, fils de n et racine d'un nouveau sous-arbre construit par appel récursif de l'algorithme A sur S_i. La sortie de l'algorithme est l'arbre enraciné en n.

Examinons maintenant le principe de construction des tests employés dans un arbre de décision. Le test associé à chaque nœud de l'arbre de décision porte sur un attribut de la représentation attribut-valeur des exemples. On distingue deux types de tests :

– ceux sur les attributs discrets ;

Algorithme 1 Algorithme A d'apprentissage d'un arbre de décision

Entrée: un ensemble d'apprentissage S

1: $n = Noeud(S)$
2: **si** $Arret(S)$ **alors**
3: $n.classe = SelectionClasse(S)$
4: **sinon**
5: $n.test = SelectionTest(S)$
6: soient $v_1, \ldots, v_k$ les valeurs prises par le test $n.test$
7: **pour** $i = 1$ **à** k **faire**
8: $S_i = \{x \in S \mid n.test(x) = v_i\}$
9: $n_i = A(S_i)$
10: affecter n_i comme $i^{\text{ième}}$ fils de n
11: **fin pour**
12: **fin si**

Sortie: l'arbre de racine n

– et ceux sur les attributs continus.

Soit t^d un test sur un attribut discret d. Dans ce cas, $t^d(x)$ est la valeur de l'exemple x pour l'attribut d, notée $x.d$. L'ensemble V des valeurs prises par t_d est l'ensemble des valeurs possibles pour l'attribut discret d. Dans le cas d'un attribut continu c, il n'est pas possible de définir les tests comme dans le cas discret. En effet un attribut continu peut potentiellement prendre un nombre infini de valeurs, ce qui conduirait à un nombre infini de fils pour le nœud n (ligne 10). Pour remédier à cela, la technique la plus couramment employée consiste à créer plusieurs tests booléens, notés $t^c_1, \ldots, t^c_p$, pour l'attribut continu c. Le test $t^c_i(x)$ est vrai si $x.c < s_i$ et faux sinon. Ainsi un tel test a un nombre fini de valeurs, en l'occurrence deux, ce qui conduit, s'il est sélectionné par le critère $SelectionTest$, à un nœud ayant deux fils. Plusieurs méthodes ont été proposées pour choisir la valeur du seuil s_i. On considère souvent les points de coupure entre les différentes valeurs de c triées dans le sens croissant.

Le critère d'arrêt de l'algorithme A influe fortement sur la qualité de l'arbre obtenue. Généralement, on préfère les arbres de petite taille. La construction de l'arbre peut être suivie d'une phase d'élagage [71] afin de poursuivre la généralisation et d'éviter le sur-apprentissage.

Nous ne décrirons pas plus en détails les algorithmes d'apprentissage d'arbre de décision. Dans la suite de cette thèse, nous utilisons l'algorithme C5.0 [73], une variante commerciale du populaire C4.5 [71] et qui intègre également la technique de combinaisons de classificateurs du boosting [32, 33, 76].

2.2.3.2 Apprenants à base de moindres généralisés

La notion de moindre généralisé est introduite par [69]. Le principe de l'apprentissage à base de moindres généralisés est de produire des hypothèses en effectuant la plus petite généralisation possible à partir de certains exemples. La recherche d'hypothèses

s'effectue en allant du plus spécifique vers le plus général.

Généralement, le calcul du moindre généralisé s'effectue avec des exemples de même classe. Le moindre généralisé de deux exemples produit l'hypothèse la plus spécifique qui couvre ces deux exemples. Dans le cas d'une représentation en attribut valeur des exemples, le moindre généralisé de deux exemples est obtenu en calculant pour chaque attribut le moindre généralisé des deux valeurs prises par les exemples. Pour un attribut continu, le moindre généralisé de deux valeurs est le plus petit intervalle qui contient ces deux valeurs. Dans le cas d'un attribut discret, le moindre généralisé de deux valeurs p et m est p si $p = m$ et la valeur indéterminée ? sinon. La figure 2.2.3.2 illustre le calcul du moindre généralisé de deux exemples.

	Bec	Robe	Poids	Taille	Classe
e_1	orange	noire	69	75	pingouin
e_2	orange	grise	77	77	pingouin
mg_1	orange	?	[69,77]	[75,77]	pingouin

FIG. 2.1 – Moindre généralisé de deux exemples : mg_1 est le moindre généralisé obtenu à partir des exemples e_1 et e_2

On peut facilement montrer qu'en attribut valeur, le moindre généralisé ainsi calculé est unique. Le calcul du moindre généralisé s'applique également à un exemple et une hypothèse, ainsi qu'à deux hypothèses (voir la figure 2.2.3.2). Un des intérêts du moindre généralisé est de fournir une hypothèse compréhensible par un expert humain.

	Bec	Robe	Poids	Taille	Classe
mg_1	orange	?	[69,77]	[75,77]	pingouin
e_3	rouge	poivre et sel	72	80	pingouin
mg_2	?	?	[69,77]	[75,80]	pingouin

FIG. 2.2 – Moindre généralisé d'un exemple et d'une hypothèse : mg_2 est le moindre généralisé obtenu à partir de l'exemple e_3 et de l'hypothèse mg_1

Dans le cadre de cette thèse, nous nous intéressons à des algorithmes produisant des moindres généralisés maximalement corrects [84]. Un tel moindre généralisé est correct dans le sens où il ne couvre pas d'exemples d'une autre classe que la sienne. Il est de plus maximal si on ne peut le généraliser avec aucun exemple supplémentaire de l'ensemble d'apprentissage sans perdre la propriété précédente de correction. Une méthode de calcul d'un moindre généralisé correct est la suivante. Le calcul d'un moindre généralisé commence à partir d'un exemple *graine* que l'on tente de généraliser avec tous les exemples de la même classe, un à un dans un certain ordre. Si l'hypothèse obtenue est correcte, alors la généralisation est acceptée et le procédé itéré. Le calcul est terminé lorsqu'il est impossible de généraliser l'hypothèse courante avec un exemple supplémentaire sans qu'elle perde sa correction. Le choix de la graine et l'ordre de parcours des exemples de la même classe déterminent le moindre généralisé obtenu. C'est pourquoi les algorithmes à base de moindres généralisés maximalement corrects

effectuent plusieurs fois le calcul précédant, avec un parcours aléatoire des exemples à généraliser, pour obtenir plusieurs moindres généralisés.

Nous n'entrerons pas dans les détails algorithmiques des apprenants à base de moindres généralisés maximalement corrects que nous avons retenus : DLG et AdaBoostMG. Pour cela, nous renvoyons le lecteur, respectivement, à [85] et à [84]. Rapidement, disons que DLG est un algorithme classique de couverture séquentielle (voir [62] page 275), et AdaBoostMG une instanciation de AdaBoost [31, 34], le célèbre algorithme de boosting, avec des moindres généralisés comme apprenant faible. Ces deux algorithmes d'apprentissage produisent des classificateurs consistants (voir la section 2.2.1 à la page 35).

2.3 La Classification Supervisée appliquée à l'Extraction d'Information

La classification supervisée a été appliquée avec succès à l'extraction d'information [29, 49, 17, 60, 61, 27, 35, 36]. Le passage de l'extraction d'information à la classification supervisée repose sur l'idée simple de *classer pour extraire*. Une représentation adéquate d'un document permet de le segmenter en unités sur lesquelles portera la classification. Une information est constituée de plusieurs unités du document et elle est extraite selon la classe que le procédé de classification lui attribue. Une telle approche aboutit à un système d'extraction d'information constitué de deux phases différentes : apprentissage et extraction. Ces deux phases sont précédées du même pré-traitement de *représentation* ou *codage* des unités, ou *exemples*, considérées. La phase d'apprentissage utilise un ensemble de documents étiquetés pour produire un modèle de classification capable d'identifier les données à extraire. La phase d'extraction applique le modèle appris à des documents non étiquetés pour en extraire les données.

La transformation d'un problème d'extraction en un problème de classification nécessite de spécifier :

- la représentation des documents ;
- la définition des éléments des documents qui seront considérés comme les exemples du problème de classification ;
- le codage de ces exemples.

Ces briques de base sont communes à beaucoup d'algorithmes d'extraction d'information, qu'ils soient basés sur la classification supervisée ou pas. Cependant, elles sont souvent implicites et intimement liées à un algorithme d'apprentissage spécifique. Notre but est ici de rendre explicites ces briques de base et d'illustrer quelles sont les possibilités pour la représentation des documents, la définition des exemples et leur codage.

Une fois fixés la représentation des documents, la définition et le codage des exemples, on peut poser le problème d'extraction d'information comme un problème de classification supervisée. L'extraction consiste à classer les exemples, représentés selon le codage choisi, et à retourner ceux affectés à la classe représentant les données à extraire[2]. On

[2]cette classe sera désormais nommée *à extraire*

définit ainsi un extracteur à partir d'un procédé de classification. Le problème d'induction d'extracteur se ramène alors à un problème d'apprentissage de classification supervisée.

Afin d'illustrer la transformation du problème d'extraction d'information en un problème de classification supervisée, on se limitera dans la suite de cette section à l'extraction unaire dans les documents non structurés et à l'examen de trois systèmes d'extraction d'information basés sur la classification supervisée : BWI [29], ELIE [27], qui sont très proches, et PaF_{texte} [61]. Pour chacun d'entre eux, nous étudierons

- la représentation des documents ;
- la définition des exemples ;
- le codage des exemples ;
- le langage d'hypothèses ;
- l'algorithme d'extraction ;
- le choix des exemples positifs et négatifs ;
- et enfin l'algorithme d'apprentissage associé.

Le but de l'étude des ces trois systèmes est moins la comparaison de leurs performances ou leur critique que la mise en évidence d'une architecture commune permettant de concevoir un système d'extraction d'information basé sur la classification supervisée.

Précisons brièvement la notion de documents non structurés. Il s'agit essentiellement de textes en langage naturel créés manuellement, comme le corps des messages électroniques, les dépêches de presse, ou encore les appels à publication des conférences scientifiques. Bien qu'il existe des techniques d'extraction à partir de tels documents utilisant des outils de traitement du langage naturel [70], les systèmes BWI, ELIE et PaF_{texte} utilisent essentiellement l'aspect syntaxique des documents.

2.3.1 BWI

BWI [29] est le premier système d'extraction d'information utilisant les techniques d'apprentissage de la classification supervisée. BWI permet d'induire des extracteurs unaires pour les documents non structurés. Un document est représenté comme une séquence d'unités lexicales, ou *tokens*. Une information à extraire est une sous-séquence de la séquence de tokens représentant le document et elle est caractérisée par un couple de positions respectivement appelées *position début* et *position fin*, qui délimitent la séquence de tokens correspondant à cette donnée. Une position début (respectivement fin) est la position dans le document qui se trouve juste avant (respectivement après) le premier (respectivement dernier) token d'une valeur à extraire.

L'algorithme d'extraction de BWI consiste à :

- déterminer les positions début ;
- déterminer les positions fin ;
- associer correctement les débuts et les fins.

Le résultat de l'extraction est l'ensemble des séquences de tokens correspondants aux couples de positions début et fin identifiés par BWI.

Le problème d'identifier si une position est un début ou pas (respectivement une fin ou pas) est ramené à un problème de classification binaire de positions dont le but est de déterminer pour chaque position dans le document s'il s'agit d'un début ou pas

(respectivement d'une fin ou pas). La ré-association d'une position début avec la bonne position de fin se fait à l'aide d'un histogramme estimant la taille des données à extraire. Ainsi l'algorithme d'apprentissage de BWI est le suivant :

- apprendre un classificateur pour reconnaître les positions début (noté c_d) ;
- apprendre un classificateur pour reconnaître les positions fin (noté c_f) ;
- construire l'histogramme (noté $h_{d,f}$) des fréquences des tailles, en nombre de tokens, des données à extraire permettant d'associer les positions débuts et fins .

2.3.1.1 Représentation des documents

Dans BWI, un document est représenté par une séquence d'unités lexicales ou tokens. Les types de tokens utilisés sont :

- les séquences de caractères alphanumériques ;
- les caractères de ponctuation ;
- le caractère *'retour chariot'*.

Toute autre séquence de caractères est ignorée.

2.3.1.2 Définition et codage des exemples

Dans BWI, un exemple est une position dans le document, entre deux tokens adjacents.

Le codage des exemples est très simple et identique pour les positions débuts et les positions fins. Chaque position est représentée par deux ensembles : l'ensemble des tokens se trouvant à sa gauche et celui des tokens se trouvant à sa droite.

2.3.1.3 Langage d'hypothèses

Les hypothèses de BWI sont des *règles* à base de motifs. Une règle, ou classificateur élémentaire, prend en entrée une position et détermine s'il s'agit ou pas d'une position début (ou fin) selon les tokens présents de part et d'autre de cette position. Une règle est constituée de deux motifs : un motif gauche et un motif droit. Chaque motif est une séquence de tokens ou de wildcards. Un wildcard est une généralisation d'un ensemble de tokens. C'est un motif globalisant, comme les jokers `*` et `?` des shells Unix. Par exemple, le wildcard `<Any>` dénote n'importe quel token, tandis que le wildcard `<ANum>` désigne l'ensemble des tokens alphanumériques. Un classificateur élémentaire pour les positions débuts classe une position comme étant un début si son motif gauche correspond à la séquence de tokens à gauche de cette position et si le motif droit correspond aux tokens à droite de la position.

2.3.1.4 Choix des exemples positifs et négatifs

Dans BWI, lorsque l'on cherche à identifier les positions débuts (respectivement fins), les exemples positifs sont les positions annotées comme étant des débuts (respectivement des fins) et les exemples négatifs sont toutes les autres positions.

2.3.1.5 Algorithme d'extraction

L'algorithme d'extraction de BWI se décompose ainsi :
- classer les positions à l'aide du classificateur de position début c_d ;
- classer les positions à l'aide du classificateur de position fin c_f ;
- associer chaque position classée comme début à une position classée comme fin à l'aide de $h_{d,f}$.

Toutes les positions des documents d'entrée sont des positions candidates pour l'extraction auxquelles les classificateurs c_d et c_f sont appliqués. Les classificateurs c_d et c_f sont tous les deux constitués de classificateurs élémentaires combinés par la technique du boosting [34].

Après les étapes de classification à l'aide de c_d et c_f, on obtient deux ensembles : celui des positions classées comme début et celui des positions classées comme fins, respectivement notés D et F. La ré-association d'une position début avec une position fin se fait à l'aide de l'histogramme $h_{d,f}$. Pour chaque couple de positions (d, f) avec $d \in D$ et $f \in F$, avec la contrainte que d se trouve avant f dans le document, $h_{d,f}$ permet d'associer un score au couple (d, f) qui est la fréquence la distance, comptée en nombre de tokens, séparant d de f. Seul le couple de score le plus élévé est retenu et extrait. Les possibilités d'extraction unaire de BWI sont ainsi très limitées car il ne peut extraire qu'une seule valeur de chaque document.

2.3.1.6 Algorithme d'apprentissage

L'algorithme d'apprentissage de BWI consiste à :
- apprendre le classificateur c_d ;
- apprendre le classificateur c_f ;
- construire l'histogramme $h_{d,f}$ d'après les documents d'apprentissage.

L'apprentissage des classificateurs c_d et c_f est réalisé de la même manière en utilisant l'algorithme de boosting AdaBoost [34, 77]. Le principe de cet algorithme est de produire un classificateur par combinaison de classificateurs élémentaires, obtenus par un autre algorithme d'apprentissage supervisé, dit *apprenant faible*. AdaBoost répète plusieurs apprentissage en faisant varier la pondération des exemples d'apprentissage à chaque étape. Chaque itération d'AdaBoost incite l'apprenant faible à se concentrer sur les exemples mal classés lors de l'étape précédente.

Dans BWI l'apprentissage des classificateurs élémentaires, constituant c_d et c_f, est assuré à chaque étape de boosting par l'apprenant faible LearnDetector. L'algorithme LearnDetector produit un classificateur élémentaire en construisant les motifs gauche et droit du classificateur. La construction de chaque motif se fait de manière heuristique (pour plus de détails voir [29]).

Chaque étape d'apprentissage nécessite la constitution de l'ensemble d'apprentissage. Dans BWI, pour l'apprentissage du classificateur c_d cet ensemble est construit de la manière suivante :
- les exemples positifs sont les positions étiquetées comme des débuts ;
- les exemples négatifs sont toutes les autres positions.

La construction de $h_{d,f}$ se résume à compter la fréquence d'apparation des tailles des données à extraire rencontrées dans l'ensemble d'apprentissage.

2.3.2 ELIE

ELIE [27] permet d'induire des extracteurs unaires pour les documents non structurés. Comme dans BWI, un document est représenté comme une séquence de tokens et une information à extraire est une sous-séquence de tokens. BWI et ELIE diffèrent sur la manière d'identifier une information. BWI utilise le couple des positions (début,fin) qui se trouvent respectivement avant et après la sous-séquence de tokens représentant l'information, tandis que ELIE identifie une sous-séquence par le couple de tokens (*token début*,*token fin*), où *token début* (respectivement *token fin*) est le premier (respectivement dernier) token de cette séquence. Cette différence de modélisation est cependant minime.

L'algorithme d'extraction d'ELIE est similaire à celui de BWI :
- déterminer les tokens début ;
- déterminer les tokens fin ;
- associer correctement les débuts et les fins.

Le problème d'identifier si un token est un début ou pas (respectivement une fin ou pas) est ramené à un problème de classification binaire de tokens. La ré-association d'un token début avec le bon token fin se fait à l'aide d'un histogramme estimant la taille des données à extraire. Encore une fois, l'algorithme d'apprentissage d'ELIE est proche de celui de BWI :
- apprendre un modèle pour classer les tokens débuts (noté m_d) ;
- apprendre un modèle pour classer les tokens fins (noté m_f) ;
- construire l'histogramme permettant d'associer les tokens débuts et fins (noté $h_{d,f}$).

La différence principale entre BWI et ELIE concerne la manière d'apprendre les modèles de classification pour les débuts et les fins. ELIE utilise un algorithme en deux étapes pour apprendre les modèles m_d et m_f. La première étape d'apprentissage consiste à apprendre à identifier les tokens débuts et les tokens fins ; tandis que la seconde consiste à apprendre à détecter le token fin (respectivement début) d'un fragment à extraire connaissant son token début (respectivement fin). Enfin l'histogramme $h_{d,f}$ est estimé sur les documents d'apprentissage.

2.3.2.1 Représentation des documents

Dans ELIE, un token est soit :
- une séquence de caractères alphanumériques ;
- un caractère de ponctuation.

2.3.2.2 Définition et codage des exemples

ELIE s'intéresse aux *tokens*. Une donnée à extraire est repérée par le premier et le dernier tokens, respectivement appelés *token début* et *token fin*, de la séquence de tokens correspondant à cette donnée.

Le codage d'un token fait appel à un codage en attribut valeur et est constitué de plusieurs groupes d'attributs. Le codage d'un token t consiste à décrire les $2w + 1$ tokens compris à l'intérieur d'une fenêtre centrée sur t. Chaque token est décrit par :

- la valeur du token ;
- l'étiquette syntaxique affectée au token par le taggeur de Brill [8], comme par exemple adjectif ou nom commun ;
- le type sémantique du token (obtenu en recherchant la valeur du token dans un ensemble de dictionnaires comprenant une liste de prénoms et de nosm de famille, une liste de noms de villes et de pays,*etc*) ;
- des attributs portant sur le type du token (alphabétique, numérique, ponctuation,*etc*) et sur sa casse de caractères (première lettre en majuscule, toutes les lettres en majuscule,*etc*).

2.3.2.3 Langage d'hypothèses

Le modèle m_d (respectivement m_f) est composé de deux classificateurs c_d^1 et c_d^2 (respectivement c_f^1 et c_f^2). Les quatre classificateurs c_d^1, c_f^1,c_d^2, et c_f^2 intervenants dans ELIE sont des machines à vecteurs de support dont l'apprentissage est assuré par l'algorithme SMO de WEKA.

2.3.2.4 Algorithme d'extraction

L'architecture globale de l'algorithme d'extraction d'ELIE est la suivante :
- classer les tokens débuts à l'aide du modèle m_d ;
- classer les tokens fins à l'aide du modèle m_f ;
- associer chaque token classé comme début à un token classé comme fin à l'aide de $h_{d,f}$.

Contrairement à BWI, les prédictions des modèles m_d et m_f ne sont pas indépendantes mais combinées entre elles au sein d'un algorithme d'extraction en deux phases. Au cours de la première, le classificateur c_d^1 (respectivement c_f^1) classe tous les tokens des documents d'entrée comme début (respectivement fin) ou pas. Lors de la seconde étape, le classificateur c_f^2 (respectivement c_d^2) est appliqué aux tokens se trouvant dans le voisinage des tokens prédits comme début (respectivement fin) par le classificateur c_d^1 (respectivement c_f^1). La notion de voisinage est définie par un paramètre k. Le but de cette étape est de déterminer le token fin d'un fragment à extraire connaissant son token début et inversement. Enfin les tokens débuts et fins sont ré-associer avec l'histogramme $h_{d,f}$.

2.3.2.5 Choix des exemples positifs et négatifs

Dans ELIE le choix des exemples positifs et négatifs est intimement lié à l'algorithme d'apprentissage. C'est pour cela que ce choix est décrit dans la section suivante en même temps que l'algorithme d'apprentissage.

2.3.2.6 Algorithme d'apprentissage

L'algorithme d'apprentissage d'ELIE consiste à :
- apprendre le modèle m_d ;
- apprendre le modèle m_f ;

– construire l'histogramme $h_{d,f}$ d'après les documents d'apprentissage.

L'apprentissage des modèles m_d et m_f se fait à l'aide d'un algorithme en deux phases, calqué sur l'algorithme d'extraction. Pour la première phase, tous les tokens des documents d'apprentissage servent d'exemples pour apprendre les classificateurs c_d^1 et c_f^1. Pour l'apprentissage de c_d^1 (respectivement c_f^1), les exemples positifs sont les tokens qui débutent (respectivement terminent) une séquence à extraire et les exemples négatifs tous les autres tokens. Au cours de la seconde phase, l'apprentissage des classificateurs c_d^2 et c_f^2 seul un sous-ensemble des exemples est utilisé. Pour c_d^2 les exemples sont les tokens se trouvant à une distance donnée avant un token de fin (*i.e.* un exemple positif pour c_f^1). Un exemple est positif s'il s'agit d'un token début, et négatif sinon. Par similarité, les exemples pour c_f^2 sont les tokens se trouvant à une distance donnée après un token de début (*i.e.* un exemple positif pour c_d^1).

2.3.3 PAF$_{texte}$

PaF_{texte} [61] est un système d'extraction d'information inspiré de BWI. Il s'agit de la première contribution personnelle de cette thèse. PaF_{texte} permet d'induire des extracteurs unaire pour les documents non structurés. Comme dans BWI, un document est représenté comme une séquence de tokens et une information à extraire est une sous-séquence de tokens Elle est identifiable par le couple de positions (*position début*,*position fin*) qui la délimite.

Contrairement à BWI, PaF_{texte} manipule directement les couples de positions. Ainsi l'algorithme d'extraction de PaF_{texte} est simple car il fait appel à un seul classificateur c qui prédit si un couple de positions correspond à une information à extraire. Il n'est donc plus nécessaire de ré-associer début et fin. La contrepartie est que le nombre de couples de positions candidats à l'extraction est quadratique en fonction de la taille du texte à traiter.

L'algorithme d'apprentissage de PaF_{texte} consiste à apprendre le classificateur de couples de positions c. Cependant, comme pour l'extraction, le nombre de ces couples est quadratique en la taille des documents d'apprentissage. Ainsi le déséquilibre entre les exemples positifs et négatifs s'accentue. C'est pourquoi un procédé de sélection des exemples négatifs est utilisé.

2.3.3.1 Représentation des documents

Comme BWI, PaF_{texte} représente un document comme une séquence de tokens. Les tokens utilisés sont les suivants :

– `Alpha` pour les séquences de caractères alphabétiques ;
– `Ponct` pour les symboles de ponctuation ;
– `Num` pour les séquences de caractères numériques ;
– le caractère *'retour chariot'*.

2.3.3.2 Définition et codage des exemples

Dans PaF_{texte} un exemple est un couple de positions. Ainsi une donnée à extraire est repérée par le couple de (*séparateur début*,*séparateur fin*) qui délimitent la séquence de tokens correspondant à cette donnée.

Le codage choisi porte sur des couples de séparateurs. L'information portée par un couple de séparateurs est plus riche que celle du codage par séparateurs indépendants. En effet le codage de couples de séparateurs permet d'utiliser des informations sur la relation entre le séparateur et le séparateur fin.

Le codage d'un exemple est un vecteur d'attributs. Soit (d, f) un couple de séparateurs début (d) et fin (f). La représentation du couple (d, f) comporte :

- des attributs décrivant d ;
- des attributs décrivant f ;
- des attributs portant sur d et f à la fois.

Les séparateurs début et fin sont codés de la même manière. Chaque séparateur est représenté par les tokens se situant à sa gauche et à sa droite, dans une fenêtre de taille T. Chaque token à l'intérieur de ladite fenêtre est codé par un à plusieurs groupes d'attributs.

Nous présentons les autres groupes utilisés. Le premier d'entre eux consiste à représenter chaque token de la fenêtre par un seul attribut indiquant son type. Ainsi un couple de séparateurs est représenté par $4.T$ attributs correspondant aux T tokens à gauche et à droite des deux séparateurs du couple.

Deux groupes d'attributs sont fondés sur l'utilisation de dictionnaires.

- Le principe du groupe d'attributs *dictionnaire* est de tester si la valeur du token est présente ou non dans un certain nombre de dictionnaires. Pour ce faire, un attribut booléen par dictionnaire utilisé est introduit pour chaque token de la fenêtre. Les trois dictionnaires de BWI sont disponibles (dictionnaire des mots anglais[3], dictionnaire des prénoms et des noms de famille les plus fréquents aux États-Unis[4]), et il est possible d'utiliser un ou plusieurs d'entre eux. Par exemple, si l'on utilise seulement le dictionnaire anglais, on aura pour chaque token de la fenêtre un attribut `ISEW` qui vaut vrai si le mot est dans le dictionnaire anglais, et faux sinon.
- Le groupe *dictionnaire utilisateur* permet à l'utilisateur de définir un dictionnaire comportant des valeurs (de tokens) qui lui semblent importantes pour trouver l'information à extraire, et que l'on s'attend généralement à trouver dans son voisinage. Un attribut catégoriel est introduit par token. Si le token est dans le dictionnaire, la valeur de l'attribut est celle du token. Sinon l'attribut prend la valeur `autre`. On définit par exemple un dictionnaire `DICOtime` contenant les valeurs `Time, :, at, Date` et pour chaque token de la fenêtre un attribut catégoriel `ISinDICOtime` qui vaut la valeur du token si elle est présente et `autre` sinon.

Enfin, le groupe *position* ne porte pas sur les tokens de la fenêtre considérée mais est simplement constitué de deux attributs, l'un indiquant la position, relative à la taille

[3] `/usr/share/dict/words` sur `Unix/Linux`

[4] disponibles à http://www.census.gov/genealogy/names/

du texte, et l'autre le numéro de ligne du séparateur.

Les attributs portants sur le couple (d, f) sont :

- la distance,comptée en nombre de tokens, séparant d de f dans le texte ;
- plusieurs attributs effectuant des comptages sur les tokens encadrés par d et f, comme par exemple le nombre de tokens numériques, le nombre de majuscules, *etc* ;
- la distance et la position de (d, f) par rapport aux mots du dictionnaire utilisateur.

Aux combinaisons des différents attributs s'ajoute la variation du paramètre T (taille de la fenêtre sur les tokens). Ce paramètre a été fixé empiriquement à la valeur 5.

2.3.3.3 Langage d'hypothèses

Les hypothèses de PaF_{texte} sont des arbres de décisions.

2.3.3.4 Algorithme d'extraction

L'algorithme d'extraction de PaF_{texte} est extrêmement simple :

- les couples candidats à l'extraction sont tous les couples de positions possibles (la seule contrainte est que la position début doit se trouver avant la position fin dans le document) ;
- le classificateur de couples c préalablement appris est utilisé pour étiqueter positif ou négatif chacun des couples candidats.

Les couples reconnus comme positifs représentent les données à extraire.

2.3.3.5 Choix des exemples positifs et négatifs

Dans PaF_{texte}, un exemple positif est un couple de positions délimitant une donnée à extraire. Considérer tous les autres couples de positions comme exemples négatifs conduirait à un nombre d'exemples trop important. Afin de réduire ce nombre une information supplémentaire est utilisée. Pour chaque problème d'extraction d'information, on observe la plus grande longueur (en nombre de tokens) de la donnée et seuls les couples de séparateurs délimitant des sous-séquences d'au plus cette longueur sont considérés comme des exemples du problème d'apprentissage. Ainsi, tous les exemples positifs sont dans la base d'exemples, mais le nombre d'exemples négatifs diminue. Cela revient à faire l'hypothèse que la taille d'une donnée à extraire est bornée et que l'on connaît cette borne à travers les exemples dont nous disposons.

2.3.3.6 Algorithme d'apprentissage

L'algorithme d'apprentissage consiste à apprendre un classificateur de couples c à partir d'un ensemble de documents marqués. PaF_{texte} utilise C4.5 comme algorithme supervisé pour obtenir un arbre de décision capable de classer tout nouveau couple comme *positif* ou *négatif*.

Les exemples d'apprentissage sont des couples de séparateurs. Les exemples positifs sont les couples correspondants aux données annotées et les exemples négatifs sont obtenus par le procédé décrit dans la section 2.3.3.5. Bien que le nombre d'exemples négatifs soit ainsi considérablement réduit, les problèmes de classification restent fortement déséquilibrés. Ce fort déséquilibre existe de part la nature même du problème mais aussi à cause de notre codage des exemples qui représentent des couples. Dans ce cas, le nombre d'exemples négatifs augmente de manière quadratique avec la taille du texte (linéaire seulement si les exemples représentent des séparateurs comme dans BWI).

Pour remédier à cela, une version modifiée de C4.5 a été utilisée : celle-ci rééquilibre les classes en pondérant les exemples de telle sorte que les sommes des poids sur chaque classe soient égales.

2.4 Approche générique

L'idée principale de la transformation d'un problème d'extraction d'information en un problème de classification supervisée se résume par la devise *classer pour extraire*. Le but de cette section est de dégager l'architecture générale des systèmes BWI, PaF_{texte} et ELIE et de présenter une approche générique.

2.4.1 Choix de représentations

Cette section illustre les représentations des documents, les définitions des exemples et les codages qui sont utilisés par BWI [29], PaF_{texte} [61] et ELIE [27].

2.4.1.1 Représentation des documents

Dans BWI, PaF_{texte} et ELIE, un document est représenté par une séquence d'unités lexicales ou *tokens*. Cependant les représentations des trois systèmes diffèrent quand au choix des unités lexicales retenues. Plus généralement, la nature et la granularité d'un token sont diverses et varient d'un aspect purement syntaxique à un aspect plus sémantique :

- simple caractère ;
- suite de caractères dénotés par leur type (symboles de ponctuation, caractères numériques) ou par une expression régulière ;
- mots définis à l'aide de dictionnaires ou d'outils de traitement du langage naturel (lémmatiseur, identificateur d'entités nommées) ;
- groupe de mots ou phrase issus également d'une approche du langage naturel.

2.4.1.2 Nature et codage des exemples

Une fois la représentation des documents fixée, plusieurs définitions des exemples sont envisageables. Il existe également un panel de codages pour les représenter.

Représenter un document comme une séquence de tokens incite naturellement à considérer les tokens comme des exemples. C'est le choix fait par ELIE. Une donnée

à extraire est rarement limitée à un seul token : il s'agit d'une sous-séquence de la séquence de tokens du document. Pour caractériser une sous-séquence à extraire, ELIE distingue deux types de tokens :

– le token *début* qui est le premier token de cette sous-séquence ;
– et le token *fin* qui en est le dernier.

BWI et PaF_{texte} s'intéressent quant à eux aux *séparateurs*. Un séparateur est une position intermédiaire entre deux tokens adjacents. Ainsi une donnée à extraire est repérée par le couple de séparateurs, respectivement appelés *séparateur début* et *séparateur fin*, qui délimitent la séquence de tokens correspondant à cette donnée.

Même s'ils diffèrent sur la définition des exemples, le point commun des systèmes BWI, PaF_{texte} et ELIE est de représenter les exemples par un codage qui porte à la fois sur l'exemple lui-même et sur son contexte dans le document qui le contient.

2.4.2 Algorithme générique d'extraction

Une fois la tâche d'extraction reformulée comme un problème de classification supervisée, on peut proposer un algorithme générique d'extraction à base d'un procédé de classification. Dans cette section, nous présentons un algorithme générique d'extraction. On met ainsi en évidence les grandes étapes pour construire un extracteur à partir d'un classificateur.

L'algorithme 2 est un algorithme d'extraction générique construit à partir d'un classificateur binaire. Il prend en entrée un document d et un classificateur d'exemples c, obtenu lors de l'induction grâce à l'algorithme 3 (abordé dans la section suivante) et fournit en sortie l'ensemble des exemples extraits de d. Cet algorithme est basé sur quatre fonctions :

– *RepresentationDocument* qui calcule la représentation du document d ;
– *ExemplesCandidats* qui sélectionne les exemples candidats à l'extraction à partir de la représentation de d ;
– *RepresentationExemple* qui retourne la représentation d'un exemple ;
– *FiltrageExemples* qui filtre les exemples extraits.

Une fois le document d en entrée représenté à l'aide de la fonction *RepresentationDocument* (ligne 1), les exemples candidats pour l'extraction sont sélectionnés (à la ligne 2) d'après la représentation r_d de d à l'aide de la fonction *ExemplesCandidats*. Le classificateur c est alors appliqué aux exemples candidats, leur représentation étant calculée avec la fonction *RepresentationExemple* (ligne 3). Les exemples extraits sont ceux classés comme positif, *i.e.* dans la classe +1. Enfin les candidats extraits sont filtrés par la fonction *FiltrageExemples* (ligne 4), selon un critère de filtrage. Par exemple, si l'on sait qu'il y a exactement un exemple à extraire, il faut en choisir un parmi tous ceux extraits. Pour ce faire, on pourra choisir l'exemple pour lequel la valeur de confiance. C'est ce qui est fait dans BWI, le critère de sélection étant la plus grande fréquence d'apparition de la taille de la valeur à extraire du classificateur c est la plus grande. Le critère de filtrage est déterminé d'après la nature de la tâche d'extraction d'information. Dans certains cas, la fonction *FiltrageExemples* est la fonction identité, c'est à dire qu'il n'y pas de filtrage et que tous les candidats extraits sont conservés. C'est le cas de PaF_{texte} et ELIE. La sortie de l'algorithme d'extraction est l'ensemble des exemples filtrés.

Algorithme 2 Algorithme générique d'extraction

Entrée: un document d, un classificateur c

1: $r_d = RepresentationDocument(d)$
2: $S = ExemplesCandidats(r_d)$
3: $S^+ = \{x \in S \mid c(RepresentationExemple(x)) = +1\}$
4: $S^+ = FiltrageExemples(S^+)$

Sortie: S^+

Algorithme 3 Algorithme générique d'induction

Entrée: un ensemble D de documents annotés

1: $R = \bigcup_{d \in D} \{RepresentationDocument(d)\}$
2: $S^+ = \bigcup_{r \in R} \{ExemplesPositifs(r)\}$
3: $S^- = \bigcup_{r \in R} \{ExemplesNegatifs(r, ExemplesPositifs(r))\}$
4: $c = A(\bigcup_{x \in S^+} RepresentationExemple(x), \bigcup_{x \in S^-} RepresentationExemple(x))$

Sortie: c

2.4.3 Algorithme générique d'induction d'extracteur

Nous présentons ici un algorithme générique d'induction d'un extracteur fonctionnant selon les principes de l'algorithme d'extraction 2.

L'algorithme générique d'induction 3 prend en entrée un ensemble D de documents complètement annotés, *i.e.* toutes les informations à extraire sont étiquetées. Ces annotations définissent la tâche d'extraction d'information. Cet algorithme retourne en sortie un classificateur c, qui est l'entrée de l'algorithme générique d'extraction 2. Tout comme dans ce dernier, la première étape de l'algorithme 3 consiste à représenter les documents à l'aide la même fonction de représentation (la fonction *RepresentationDocument* de la ligne 1). R est l'ensemble des représentations des documents de D. Ensuite les exemples positifs sont obtenus par la fonction $ExemplesPositifs$ à partir de R (à la ligne 2). Un exemple positif correspond à une information annotée comme étant à extraire. Les entités étiquetées et leur étiquetage sont préservées par la fonction de représentation.

Comme l'extraction d'information a été ramenée à un problème de classification supervisée binaire, il faut également des exemples négatifs. Ceux-ci sont obtenus par la fonction $ExemplesNegatifs$. Les documents étant complètement étiquetés, tous les exemples non annotés sont des exemples négatifs. Cependant, leur nombre peut être très grand. En effet, pour une tâche d'extraction fixée, parmi les informations présentes dans un document, généralement seule une très petite quantité d'entre elles sont à extraire. Ce déséquilibre naturel se retrouvera en classification supervisée. Il se peut même qu'il soit accentué considérablement. Par exemple, lorsque les exemples sont des couples de tokens, le nombre d'exemples positifs est le nombre de données à extraire. Par contre, comme les documents sont complètement annotés, tous les couples non annotés ne sont pas à extraire. Ce sont donc des exemples négatifs, dont le nombre est quadratique en la taille du document (calculée en nombre de tokens). Il est souvent préférable de limiter le nombre d'exemples négatifs, à la fois pour des raisons de complexité, mais aussi pour faciliter le travail de l'algorithme d'apprentissage. Par exemple, dans le cas des couples,

on peut considérer comme exemples négatifs seulement les couples (p, d) tels que d est après p et tels que (p, d) n'est pas un exemple positif. On retrouve ici un procédé de calcul similaire à celui des exemples candidats de l'algorithme d'extraction de couples à un seul classificateur (décrit dans la section précédente). Si l'on connaît une borne M sur la taille de la donnée à extraire, on peut même considérer comme exemples négatifs les couples (p, d) pour lesquels p et d sont séparés par au plus M tokens [60, 61]. Comme on vient de le voir, la fonction $ExemplesNegatifs$ sélectionne un sous-ensemble des exemples négatifs, d'une manière assez proche de la sélection des exemples candidats de l'algorithme 2. Cette sélection peut se faire d'après la représentation du document, mais aussi d'après les exemples positifs (ligne 3). La fonction $ExemplesNegatifs$ doit s'assurer que parmi les exemples sélectionnés comme exemples négatifs, aucun d'eux n'est positif. En effet, cela introduirait des ambiguïtés dans l'ensemble d'apprentissage, qui deviendrait ainsi inconsistant, car un même exemple aurait deux classes différentes. Avec l'ensemble des exemples positifs, la fonction $ExemplesNegatifs$ peut s'assurer de la consistance de l'ensemble d'apprentissage lors de la sélection des exemples négatifs.

Les exemples positifs S^+ et négatifs S^- constituent l'ensemble d'apprentissage de l'apprenant de base A (ligne 4), qui renvoie le classificateur c appris. Ce classificateur sera l'entrée de l'algorithme générique d'extraction 2.

Les grandes étapes de l'algorithme d'induction restent les mêmes, à savoir :
- représentation du document ;
- accès aux exemples positifs ;
- sélection des exemples négatifs ;
- puis apprentissage.

C'est sur ce schéma que sera construit l'algorithme d'induction d'extracteur de la section 4.7.

2.5 Conclusion

Dans ce chapitre, nous avons présenté les notions de la classification supervisée utilisées dans cette thèse. Ensuite nous avons vu comment transformer un problème d'extraction d'information en un problème de classification supervisée, dans le cas de l'extraction unaire dans les documents non structurés. Nous avons mis en évidence les grandes étapes intervenant au cours de cette transformation à travers l'étude de trois systèmes d'induction d'extracteurs basés sur la classification supervisée. Il s'agit essentiellement de choisir une représentation des documents et de définir la nature des exemples et leur codage. On peut alors décrire le fonctionnement d'un extracteur selon un procédé de classification et proposer l'algorithme d'induction associé. Dans ce chapitre, nous avons dégagé de l'analyse de l'existant des algorithmes génériques d'extraction et d'induction d'extracteurs basés sur la classification supervisée. Ils sont la base de notre approche de l'extraction d'information n-aire pour les documents arborescents, à laquelle le reste de cette thèse est dédiée et que nous présentons en détail dans le chapitre 4. Le chapitre suivant, quand à lui, étudie les diverses manières de stocker des n-uplets dans un arbre.

Chapitre 3

Extraction n-aire dans les structures arborescentes

3.1 Introduction

À partir de ce chapitre, nous nous focalisons sur l'extraction d'information n-aire dans les documents arborescents. Après avoir défini la tâche d'extraction d'information n-aire qui nous intéresse, nous étudions ensuite les différentes manières de stocker les n-uplets d'une relation n-aire dans une structure arborescente. À partir d'observations de plusieurs sites Web, nous avons mis en évidence cinq organisations que l'on rencontre fréquemment au sein des documents `HTML`. Il s'agit de la contribution majeure de ce chapitre. Enfin nous discutons de l'expressivité des systèmes d'extraction d'information n-aire présentés dans la section A, mais aussi de celle de deux systèmes non supervisés. Le but est ici double. D'une part il s'agit de montrer, empiriquement, qu'une approche non supervisée ne répond pas correctement à la tâche d'extraction d'information telle qu'on l'envisage dans cette thèse. D'autre part, la mise en valeur des cinq organisations de base des données n-aires permet de regarder l'existant d'un œil nouveau pour en montrer les limites. La capacité à traiter les différentes organisations des données des systèmes supervisés est examinée. Cette étude montre que la plupart d'entre eux sont spécifiques à un type d'organisation de l'information et incapables de traiter les cinq cas que nous avons identifiés.

3.2 Différentes tâches d'extraction n-aire

La diversité des tâches d'extraction n-aire dépend de deux facteurs :

- la répartition des n-uplets dans un ensemble de documents `HTML` ;
- la granularité variable des composantes à extraire.

3.2.1 Répartition des n-uplets

Avant même d'envisager l'organisation des n-uplets dans une structure arborescente, il faut discuter de leur répartition au sein d'un ensemble de documents. On distingue deux cas. Le premier est le plus simple : chaque n-uplet est contenu dans un seul document et toutes ses composantes se trouvent dans ce document. Dans le second cas, les valeurs des composantes d'un n-uplet sont réparties sur plusieurs documents. Les sites marchands présentent souvent les informations sur les produits de cette manière : une page donne la liste des produits, avec des caractéristiques générales comme son nom et son prix, tandis que les informations détaillées d'un article se trouvent sur une autre page, à laquelle on accède par un lien depuis la première. Une telle répartition des composantes des n-uplets impose la capacité de naviguer entre les différentes pages impliquées.

3.2.2 Granularité d'une composante

Dans les documents arborescents, les données sont contenues dans les feuilles textuelles de l'arbre et la structure de l'arbre représente la structure logique du document. La valeur d'une composante peut être :

- exactement le contenu d'une feuille ;
- une partie d'une feuille textuelle ;
- répartie sur plusieurs feuilles.

3.2.2.1 Extraction d'une feuille (cas de base)

Dans le cas le plus simple, une valeur à extraire est exactement le contenu d'une feuille texte. Ce cas sera désigné par la suite sous le nom de *cas de base*. C'est celui qui est considéré par SQUIRREL$_n$ et que nous envisageons dans [36, 37].

Une information à extraire peut aussi être la valeur d'un attribut `HTML`. Dans l'arbre, un attribut est représenté par un nœud, dont le label est le nom de cet attribut et qui a une unique feuille texte dont le contenu est la valeur prise par l'attribut. Par exemple il peut s'agir d'extraire l'ensemble des adresses mails d'une page d'un annuaire électronique, ou encore tous les liens hypertextes pointant sur les pages personnelles des personnes figurant dans cet annuaire.

3.2.2.2 Extraction d'une partie du contenu d'une feuille

La valeur d'une composante peut être une sous séquence du texte d'une feuille de l'arbre représentant un document. On peut envisager deux solutions pour traiter ce cas. La première consiste à extraire, en utilisant la vue structurelle, la feuille qui contient la valeur, puis à rechercher dans le texte de cette feuille la sous séquence à extraire. Pour ce faire, on peut combiner les vues arborescente et textuelle. La seconde solution résulte de la représentation en tokens du contenu des feuilles textuelles. Chaque feuille texte est remplacée par autant de nouvelles feuilles qu'elle contient de tokens [19, 81, 82]. Si la valeur à extraire se limite à un seul token, on se retrouve à nouveau dans le cas simple

de base évoqué précédemment (à la section 3.2.2.1). Cependant cette transformation de l'arbre peut faire surgir une autre possibilité que nous examinons maintenant.

3.2.2.3 Extraction de plusieurs feuilles

Une valeur à extraire peut également correspondre à plusieurs feuilles textuelles. On envisage seulement le cas où ces feuilles sont contiguës, car dans le cas contraire on est certainement face à plusieurs valeurs à extraire et non pas à une seule. C'est le cas considéré par [19, 81, 82].

3.2.3 Bilan

Nous avons discuté de la répartition des tuples dans un ensemble de documents, ainsi que la granularité de la valeur d'une composante.

Dans la suite de cette thèse nous nous focalisons sur l'extraction de n-uplets de feuilles textuelles depuis des documents `HTML`, tels que la valeur d'une composante est exactement une feuille texte de l'arbre `HTML` et que toutes les composantes d'un même n-uplet sont stockées dans le même document.

Plus formellement, on peut définir un extracteur comme une fonction de l'ensemble des arbres T vers l'ensemble des n-uplets de feuilles textuelles $L(t)^n$, où $L(t)$ désigne l'ensemble des feuilles textes de l'arbre t. Bien que nous nous limitons aux documents `HTML`, cette définition de la tâche d'extraction d'information est également valable avec des documents `XML`.

Nous allons maintenant étudier les différentes manières de stocker de tels n-uplets dans un document `HTML`.

3.3 Stockage de n-uplets dans un arbre

Cette section expose notre étude de l'organisation de données n-aire dans une structure arborescente. Elle ne prétend pas à l'exhaustivité. Elle est fondée sur l'observation de divers sites Web, comme des sites de statistiques, des sites marchands et des sites météorologiques, ainsi que des pages `XHTML` produites par des outils de reporting (comme JasperReports). L'analyse de l'organisation des données qu'ils contiennent a mis en évidence cinq cas :

- table ;
- liste ;
- table tournée ;
- factorisation de certaines composantes ;
- table croisée.

Ces cinq organisations sont fréquentes, c'est pourquoi on les considère comme les cas *de base*. On vera que leur combinaison permet de produire des structures plus complexes.

Les cinq sections suivantes présentent les cinq organisations de base des données. Chacune d'entre elles est illustrée par une à plusieurs pages trouvées sur le Web. Pour mieux examiner les propriétés de chaque organisation, on utilise un document `HTML` plus simple, qui permet se focaliser uniquement sur l'agencement des n-uplets

dans la structure arborescente. Pour les quatre premières organisations (table, liste, table tournée et factorisation), les documents `HTML` simplifiés contiennent les mêmes données mais présentées différement. La relation n-aire cible est la relation ternaire `(Club,Season,Score)`. Pour les tables croisées, la relation n-aire considérée est `(Club1,Club2,Score)`.

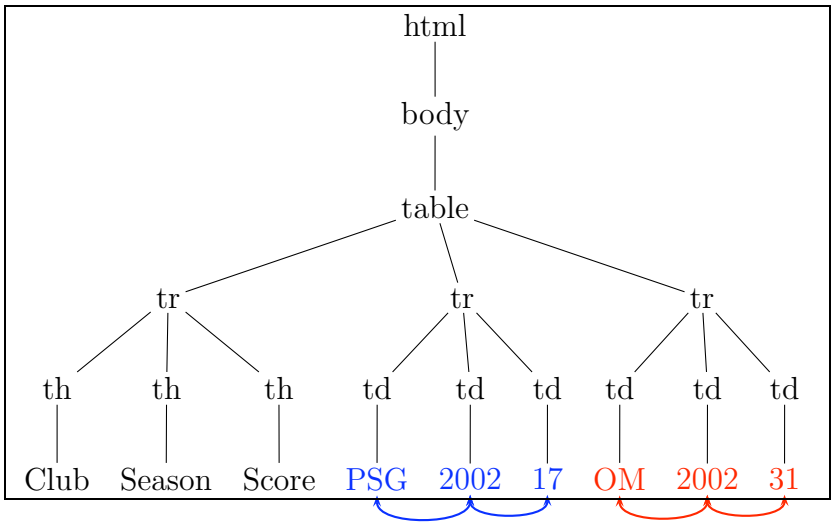
3.3.1 Organisation en table

La manière la plus naturelle d'organiser des n-uplets est de les stocker dans une table. C'est par exemple le cas des données météorologiques de la page Web de la figure 3.1, où chaque tuple est stocké dans une ligne d'une table, dont chaque colonne correspond à une composante.

FIG. 3.1 – Organisation en table : une page du site Web météorologique de la BBC

Comme l'illustre la figure 3.2, les noms des composantes se trouvent dans la première ligne et les données dans les lignes suivantes. Un n-uplet correspond à une ligne, qui est désignée par le nœud `tr`, tandis que ses valeurs sont les feuilles textes des cellules (dénotées par `td`). Chaque colonne regroupe les valeurs d'une même composante. L'ordre des colonnes étant fixe, les composantes sont toujours dans le même ordre. Si

une valeur d'un n-uplet est manquante, alors la cellule correspondante de la table est vide, mais la structure arborescente de la table est préservée. En effet une manquante dans une cellule se traduit dans l'arbre par un nœud `td` sans feuille textuelle comme fille (*i.e.* une cellule vide).



Club	Season	Score
PSG	2002	17
OM	2002	31

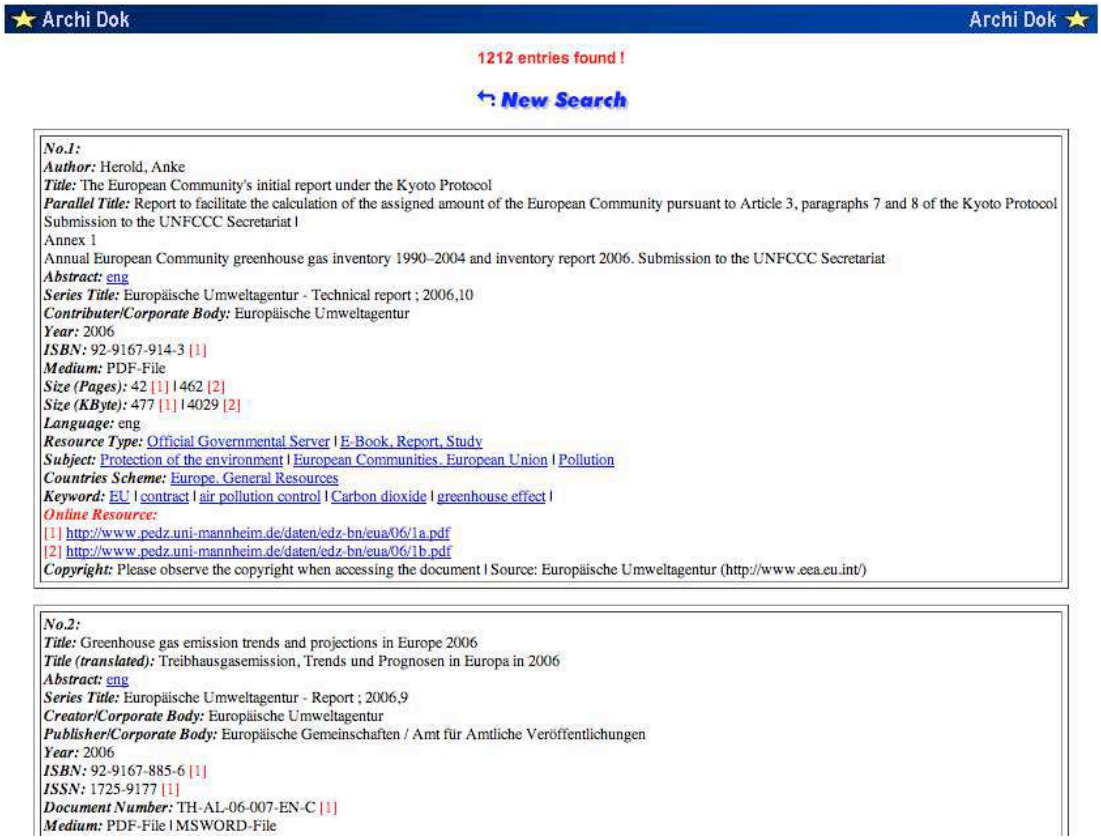
FIG. 3.2 – Table, vue arborescente en haut et rendu en bas. Les flêches indiquent les n-uplets.

Déterminer la valeur d'une composante est aisée. Par exemple, dans la figure 3.2 le contenu d'une cellule de la troisième colonne est une valeur de la composante `Score`. Dans l'arbre cela s'exprime par la règle suivante : toute feuille dont le père est étiqueté par `td` et tel que ce nœud est en troisième position dans la séquence de ses frères est un score.

On peut remarquer sur la figure 3.2 que les n-uplets ne se chevauchent pas dans l'arbre `HTML` (c'est aussi le cas dans le code source du document). Le plus petit ancêtre commun des tous les nœuds correspondant aux composantes d'un même tuple est le nœud `tr`, qui désigne une ligne. Ainsi chaque n-uplet est dans un sous arbre de racine `tr` ne contenant que les valeurs de ce n-uplet. Ses sous-arbres sont bien distincts entre eux et ne se chevauchent pas. De plus les racines de ses sous arbres sont également différentes deux à deux. Il est donc aisé de distinguer les n-uplets les uns des autres, car chaque n-uplet est contenu dans un sous arbre qui lui est propre et identifiable facilement.

3.3.2 Organisation en liste

La seconde organisation simple, qui est également très fréquente sur le Web, est celle de liste. Les n-uplets sont stockés séquentiellement dans le document, l'un après l'autre. C'est le cas de la liste de résultats de la figure 3.3. Il s'agit des résultats d'une requête posée sur le site d'archivage des documents de l'Union Européenne [1].


Archi Dok Archi Dok

1212 entries found !

New Search

No.1:
Author: Herold, Anke
Title: The European Community's initial report under the Kyoto Protocol
Parallel Title: Report to facilitate the calculation of the assigned amount of the European Community pursuant to Article 3, paragraphs 7 and 8 of the Kyoto Protocol Submission to the UNFCCC Secretariat I
Annex 1
Annual European Community greenhouse gas inventory 1990–2004 and inventory report 2006. Submission to the UNFCCC Secretariat
Abstract: eng
Series Title: Europäische Umweltagentur - Technical report ; 2006,10
Contributer/Corporate Body: Europäische Umweltagentur
Year: 2006
ISBN: 92-9167-914-3 [1]
Medium: PDF-File
Size (Pages): 42 [1] I 462 [2]
Size (KByte): 477 [1] I 4029 [2]
Language: eng
Resource Type: Official Governmental Server I E-Book, Report, Study
Subject: Protection of the environment I European Communities. European Union I Pollution
Countries Scheme: Europe. General Resources
Keyword: EU I contract I air pollution control I Carbon dioxide I greenhouse effect I
Online Resource:
[1] http://www.pedz.uni-mannheim.de/daten/edz-bn/eua/06/1a.pdf
[2] http://www.pedz.uni-mannheim.de/daten/edz-bn/eua/06/1b.pdf
Copyright: Please observe the copyright when accessing the document I Source: Europäische Umweltagentur (http://www.eea.eu.int/)

No.2:
Title: Greenhouse gas emission trends and projections in Europe 2006
Title (translated): Treibhausgasemission, Trends und Prognosen in Europa in 2006
Abstract: eng
Series Title: Europäische Umweltagentur - Report ; 2006,9
Creator/Corporate Body: Europäische Umweltagentur
Publisher/Corporate Body: Europäische Gemeinschaften / Amt für Amtliche Veröffentlichungen
Year: 2006
ISBN: 92-9167-885-6 [1]
ISSN: 1725-9177 [1]
Document Number: TH-AL-06-007-EN-C [1]
Medium: PDF-File I MSWORD-File

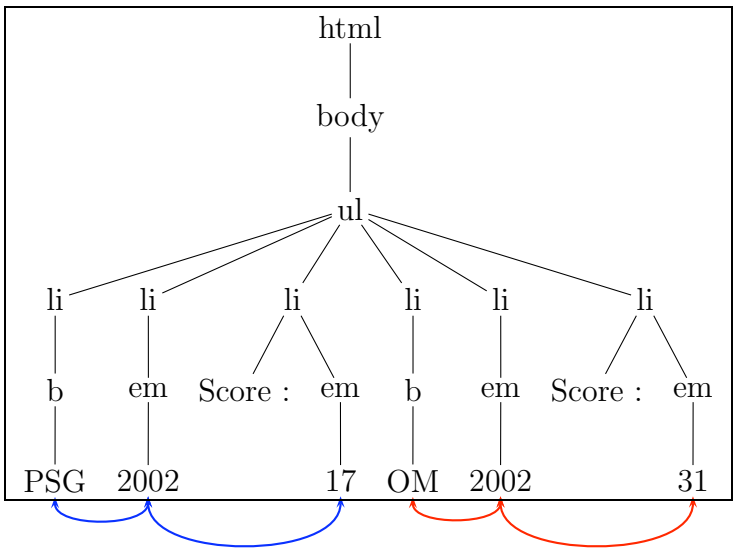
FIG. 3.3 – Organisation en liste : une page du site Archidok

On distinguera deux types de listes : les listes plates, comme celle des figures 3.4 et 3.3, et les listes imbriquées, dont la structure est proche de celle des tables [2].

L'examen de la figure 3.4 permet de constater que dans une liste plate les n-uplets ne se chevauchent pas. Comme avec les tables, chaque n-uplet est contenu au sein d'un sous arbre qui lui est propre et qui ne contient pas de valeurs provenant d'un autre tuple. Les n-uplets ne se mélangent pas dans l'arbre `HTML`. Cependant ces sous arbres ont tous la même racine. Il s'agit du nœud `ul` qui est le constructeur de la liste. L'extraction des n-uplets est donc plus délicate que précédemment, en raison d'une structure un peu plus faible et de l'absence d'une séparation nette des n-uplets.

[1] http://archidok.uni-mannheim.de/en/datenbank_en.html

[2] on peut voir une table comme une liste de lignes et une ligne comme une liste de cellules



- **PSG**
- *2002*
- Score : *17*
- **OM**
- *2002*
- Score : *31*

FIG. 3.4 – Liste

Une solution consiste à compter modulo le nombre de composantes. Dans le cas de la figure 3.4, en partant du début de la liste, un élément sur trois est une valeur de la composante `Club`. À partir du second élément, un élément sur trois est une valeur de la composante `Season`. Cependant cela n'est plus vrai lorsqu'une des valeurs d'un n-uplet est manquante. Dans ce cas, l'absence de l'élément de la liste correspondant modifie la structure de l'arbre. Un décalage des composantes ne permet plus de compter modulo l'arité de la relation. La lecture des données devient difficile, surtout si le nombre de composantes est grand. De plus d'un n-uplet à l'autre l'ordre des composantes est susceptible de changer. Séparer les tuples et distinguer les différentes composantes est alors délicat.

C'est pour cette raison qu'on rencontre généralement un délimiteur ou une valeur remarquable, qui structure l'information et facilite la lecture des données. Par exemple, dans la figure 3.4, la séquence `Score :` (le mot `Score` suivi du symbole de ponctuation `:`) est le contenu de la feuille précédant une valeur de la composante `Score`. Des variations de la forme du texte, comme l'instance ou l'accentuation sur certaines valeurs grâce à l'usage de l'italique ou du gras, participent aussi à distinguer les différentes composantes. Dans la liste de la figure 3.4, les valeurs de la composante `Club` sont en gras. Dans l'arbre leur père a comme label `b`. Repérer tous les nœuds dont le père est le nœud `b` permet de trouver tous les `Club`. Ces éléments remarquables sont des indices encore plus précieux pour les listes très pauvres structurellement et qui sont construites sans l'utilisation des balises `HTML` définissant les listes (comme `ul`, `ol` et `dl`). C'est par exemple le cas de la liste de résultats retournés par le moteur de recherche Google, dont le rendu visuel est celui d'une liste, mais qui est formatée essentiellement par des retours à la ligne (balise `br`) et des balises de mise en forme (comme `b` ou `em`).

3.3.3 Organisation en table tournée

Lorsque le nombre de composantes est plus grand que celui des n-uplets, l'utilisation d'une table ou d'une liste rend difficile la lecture des données. Dans ce cas de figure, on préfère employer une table tournée. Les données de la page, tirée du site du Census Bureau [3] de la figure 3.5 l'illustrent bien. Il s'agit pour chaque année de présenter une longue série de statistiques.

On peut constater sur la figure 3.6, que les noms des composantes sont dans la première colonne de la table et les données dans les colonnes suivantes. Un n-uplet est stocké dans une colonne et une ligne contient les différentes valeurs d'une même composante.

On peut observer sur la figure 3.6 que le plus petit sous arbre contenant chacun des triplets a pour racine le nœud `table`, constructeur de la table. Ces différents sous-arbres se chevauchent en grande partie. Au lieu de contenir uniquement les valeurs d'un seul n-uplet, comme dans le cas des tables et des listes, chacun de ces sous-arbres contient également des valeurs des autres n-uplets. La conséquence directe du chevauchement des arbres est que les n-uplets sont entrelacés, à la fois dans les vues textuelle, arborescente et feuillage d'un document. En effet, la figure 3.6 illustre bien que dans l'arbre les triplets `(PSG,2002,17)` et `(OM,2002,31)` sont entrelacés. Il en est de même si l'on

[3] http://www.census.gov

USA Statistics in Brief--Population by Sex, Age, and Region

POPULATION	2000	2002	2003	2004	2005
Resident population (1,000)	281,425	287,985	290,850	293,657	296,410
Male (1,000)	138,056	141,542	143,058	144,535	146,000
Female (1,000)	143,368	146,442	147,792	149,121	150,411
Under 5 years old (1,000)	19,176	19,537	19,778	20,061	20,304
5 to 17 years old (1,000)	53,119	53,325	53,269	53,198	53,166
18 to 44 years old (1,000)	112,184	112,979	113,210	113,373	113,313
45 to 64 years old (1,000)	61,954	66,555	68,640	70,693	72,838
65 years old and over (1,000)	34,992	35,589	35,952	36,333	36,790
Northeast (1,000)	53,595	54,192	54,427	54,582	54,642
Midwest (1,000)	64,395	65,097	65,410	65,694	65,972
South (1,000)	100,236	103,195	104,553	105,994	107,505
West (1,000)	63,199	65,502	66,461	67,387	68,291
Percent of population--	**2000**	**2002**	**2003**	**2004**	**2005**
Male (percent)	49.1	49.1	49.2	49.2	49.3
Female (percent)	50.9	50.9	50.8	50.8	50.7

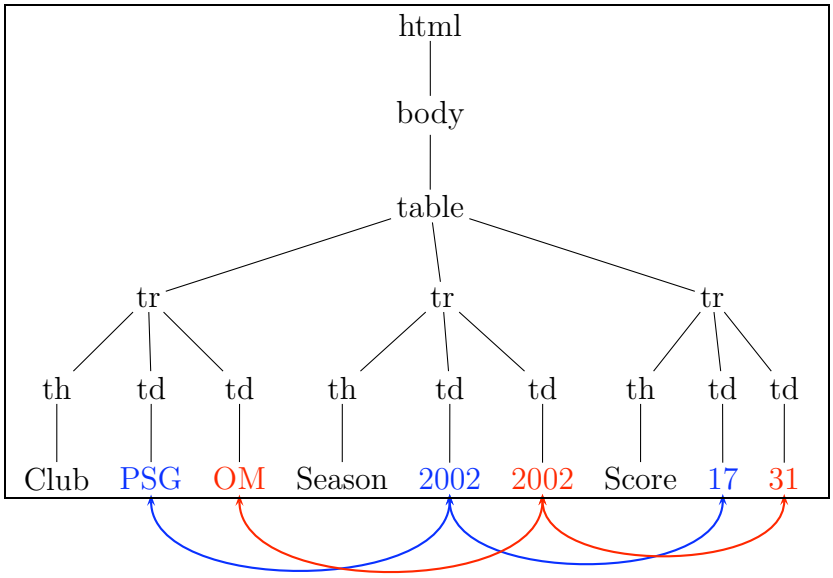
FIG. 3.5 – Organisation en table tournée : une page du site Census Bureau

considère le feuillage de l'arbre : de gauche à droite, on rencontre d'abord les noms des clubs `PSG` et `OM`, puis les saisons avec deux fois la valeur `2002` et enfin les scores `17` et `31`. Dans le feuillage d'une table classique, comme celle de la figure 3.2, on rencontre les valeurs des composantes `Club`, `Season` puis `Score` du premier triplet avant celles du second.

Identifier la valeur d'une composante est facile. Par exemple le contenu d'une cellule de la troisième ligne de la table (de la figure 3.6) est un score. Dans l'arbre cela correspond à toute feuille texte dont le grand père est un nœud `tr` qui est lui-même le troisième fils de son père (l'arrière grand père de la feuille).

Par contre, comme les n-uplets sont entrelacés, associer correctement les valeurs des composantes entre elles pour former les n-uplets est plus délicat. La structure arborescente est la seule vue sur les documents qui permet de le faire. Les valeurs d'un même n-uplet sont dans la même colonne. Cette contrainte se traduit dans l'arbre par le fait que les feuilles correspondantes se retrouvent à la même position dans l'arbre. Par exemple dans le document `HTML` de la figure 3.6, les valeurs `PSG`, `2002` et `17`, qui font partie du même triplet, sont en dessous du deuxième fils `td` de chaque nœud `tr` de la table.

Dans le cas d'une table tournée, l'extraction des n-uplets nécessite de compter des positions dans l'arbre et de mémoriser la valeur de ses positions. Or en terme de théorie des langages, le langage d'arbres correspondant n'est pas régulier lorsque la taille de la table n'est pas bornée.



Club	PSG	OM
Season	2002	2002
Score	17	31

FIG. 3.6 – Table tournée

3.3.4 Organisation avec factorisation

Pour éviter les répétitions de valeurs et raccourcir la présentation des données, on utilise souvent une structure imbriquée. Ce type d'organisation est fréquente dans les tableurs avec l'usage des valeurs pivots qui sont factorisées. La figure 3.7 présente une liste de chambres d'hôtels à réserver. Le nom de chaque hôtel apparaît une seule fois dans la liste et il est suivi de la liste de ses chambres libres.

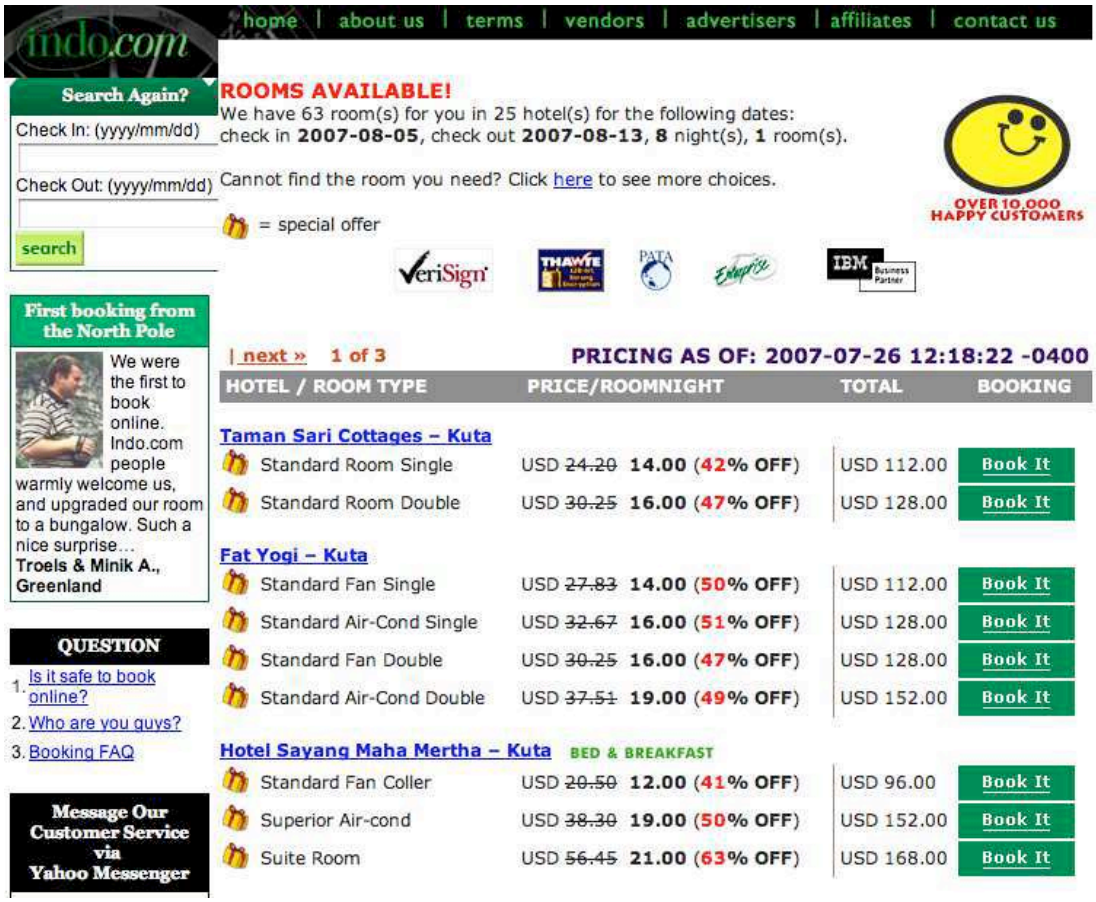
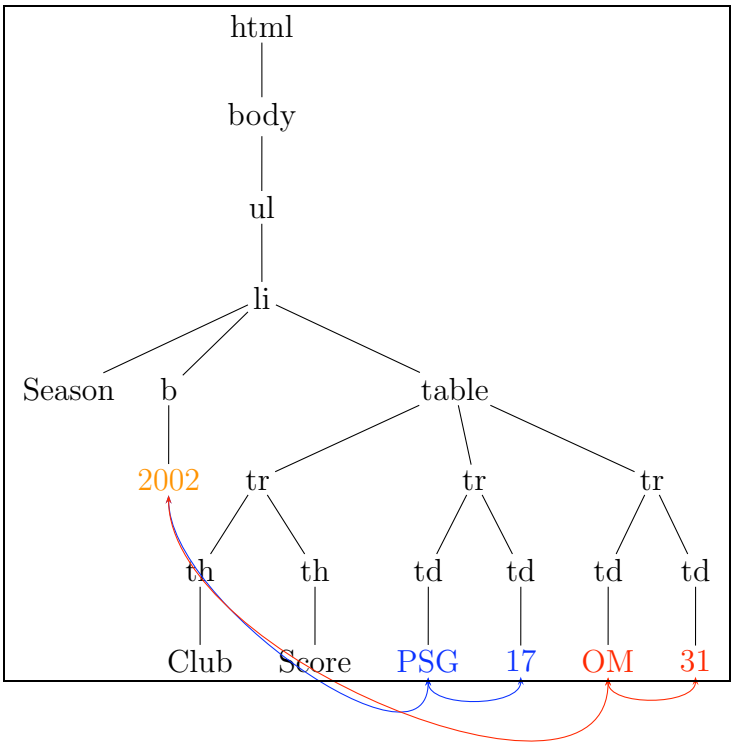


FIG. 3.7 – Organisation avec factorisation : une page d'un site de réservation en ligne de chambres d'hôtels

Les triplets `(PSG,2002,17)` et `(OM,2002,31)` partagent la même valeur pour la seconde composante. Au lieu de la répéter deux fois, comme c'est le cas dans la table de la figure 3.2, on peut la factoriser, comme l'illustre la figure 3.8. Les valeurs des composantes `Club` et `Score` sont rangées dans une table et la valeur de la composante `Season` est présente une seule fois dans le document, juste avant la table.

Sur cette même figure on peut constater que les n-uplets sont à nouveau entrelacés dans l'arbre. La valeur de la composante `Season` apparaît avant celles des autres



- Season **2002**

Club	Score
PSG	**17**
OM	**31**

FIG. 3.8 – Organisation avec valeurs factorisées. La valeur 2002 est factorisée sur les deux triplets (PSG,2002,17 et (OM,2002,31)

composantes.

3.3.5 Organisation en table croisée

Les tables croisées permettent la présentation de données multi-dimensionnelles. Elles sont d'un usage courant dans les rapports de statistiques ou les tables de distances kilométriques entre plusieurs villes ou les tables de conversion de devises.

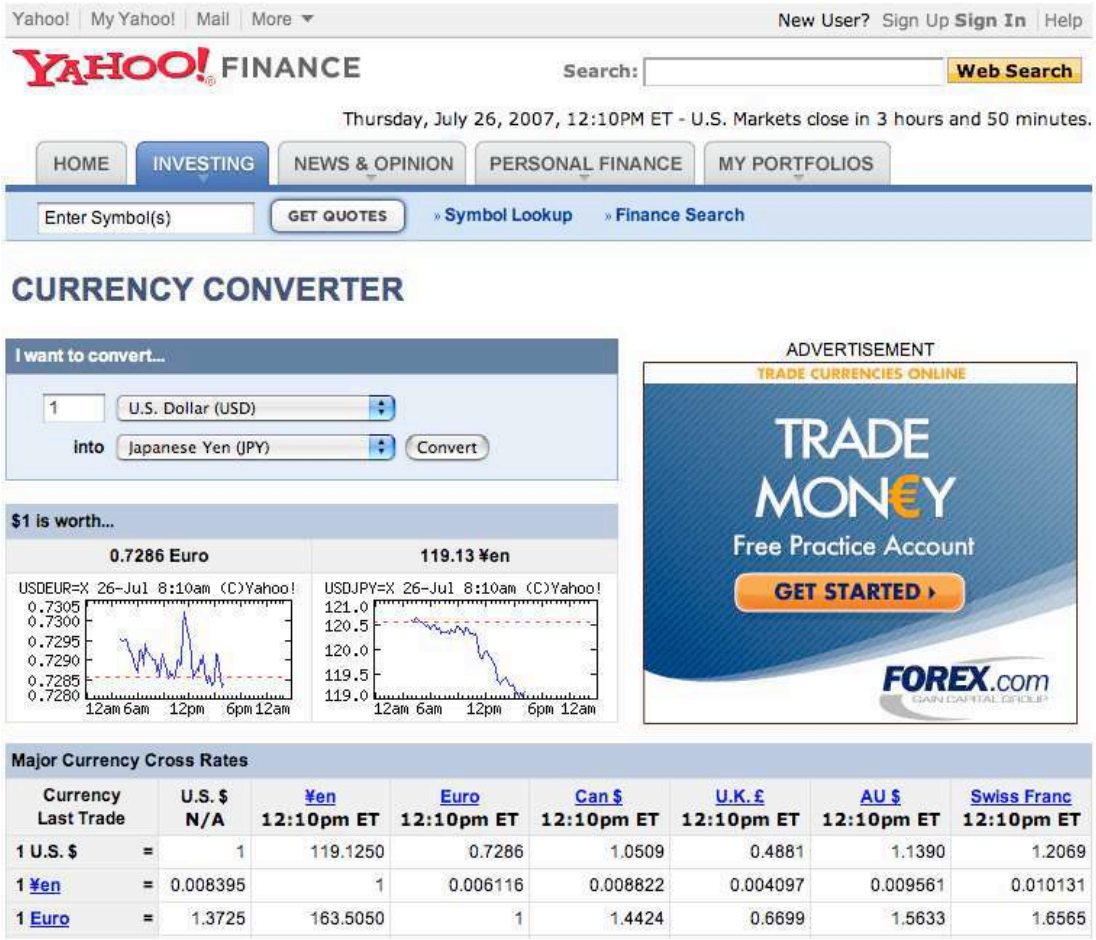
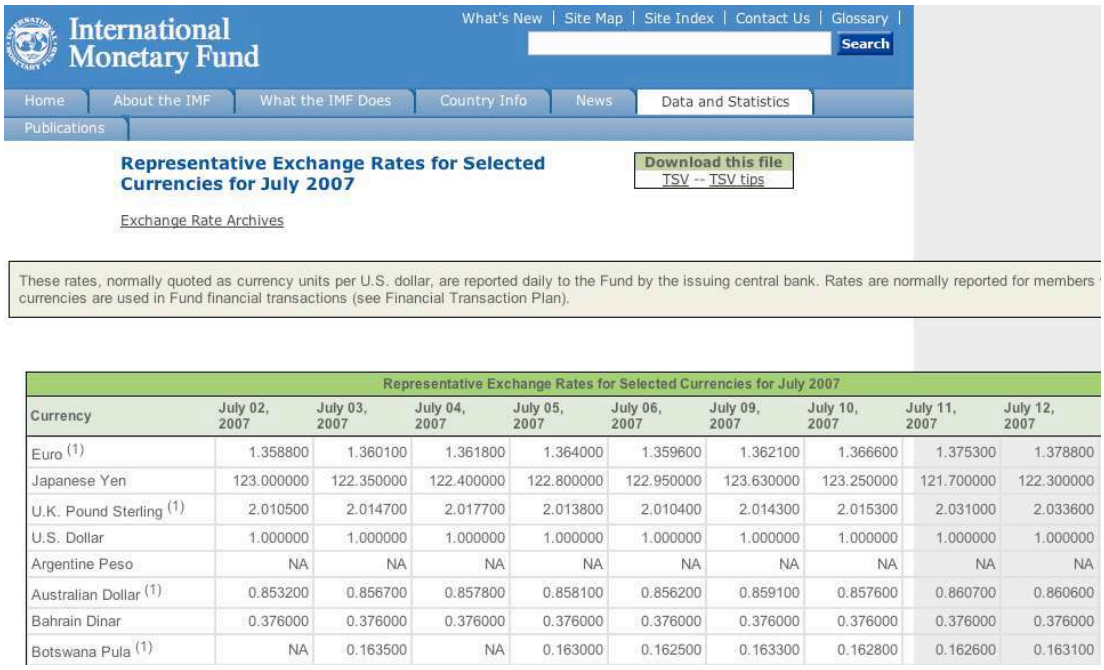


FIG. 3.9 – Organisation en table croisée : une page du site Web Yahoo Finance

La figure 3.9 présente une page Web du site Yahoo Finance [4] qui contient une table croisée de conversion de devises. Les devises apparaissent dans la première ligne et la première colonne de la table. Le taux de conversion se situe à la croisée d'une ligne et d'une colonne.

[4] http://finance.yahoo.com



Representative Exchange Rates for Selected Currencies for July 2007

Currency	July 02, 2007	July 03, 2007	July 04, 2007	July 05, 2007	July 06, 2007	July 09, 2007	July 10, 2007	July 11, 2007	July 12, 2007
Euro (1)	1.358800	1.360100	1.361800	1.364000	1.359600	1.362100	1.366600	1.375300	1.378800
Japanese Yen	123.000000	122.350000	122.400000	122.800000	122.950000	123.630000	123.250000	121.700000	122.300000
U.K. Pound Sterling (1)	2.010500	2.014700	2.017700	2.013800	2.010400	2.014300	2.015300	2.031000	2.033600
U.S. Dollar	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
Argentine Peso	NA	NA	NA	NA	NA	NA	NA	NA	NA
Australian Dollar (1)	0.853200	0.856700	0.857800	0.858100	0.856200	0.859100	0.857600	0.860700	0.860600
Bahrain Dinar	0.376000	0.376000	0.376000	0.376000	0.376000	0.376000	0.376000	0.376000	0.376000
Botswana Pula (1)	NA	0.163500	NA	0.163000	0.162500	0.163300	0.162800	0.162600	0.163100

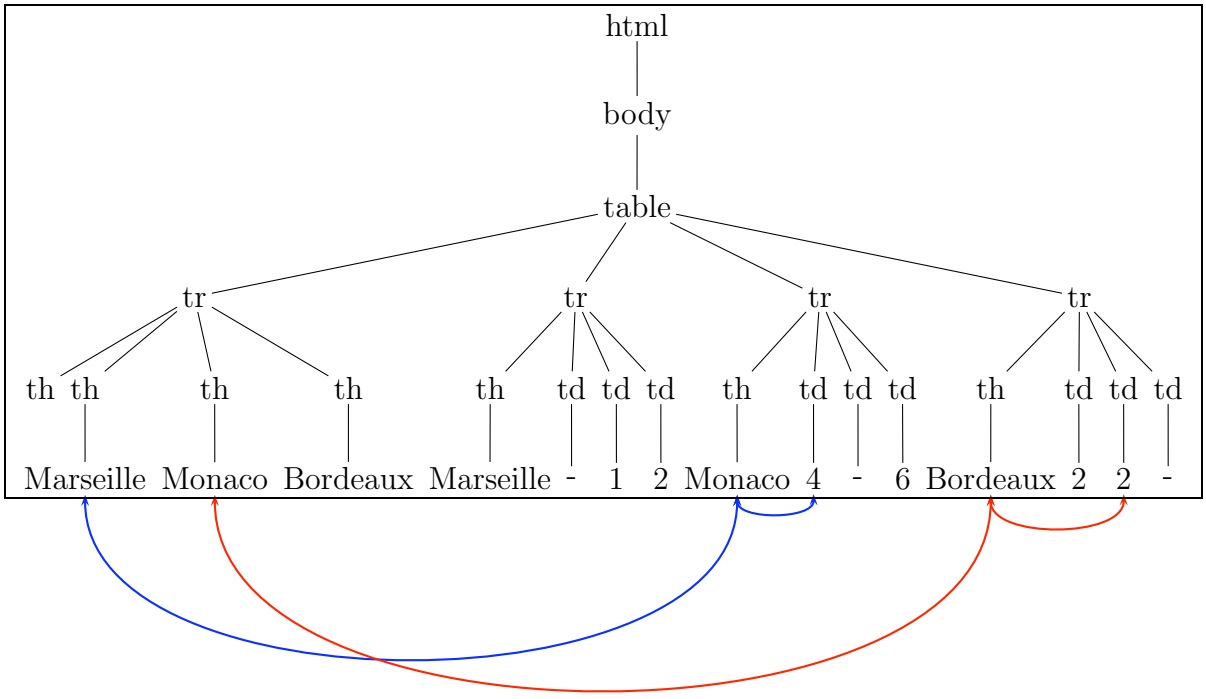
FIG. 3.10 – Organisation en table croisée : une page du site Web du FMI

La page du site du Fond Monétaire International [5] de la figure 3.10 contiennent aussi une table croisée qui présente cette fois l'évolution du taux de convertion de plusieurs devises (dans la première colonne) jour par jour au mois de juillet 2007 (première ligne).

La relation n-aire de la table croisée de la figure 3.11 est `(Club1,Club2,Score)`. Les valeurs de la composante `Club1` sont dans la première ligne de la table. Il s'agit des feuilles textes du sous-arbre dont la racine est le premier fils `tr` du nœud `table`. La première colonne, quand à elle, contient les valeurs de la composante `Club2`. Dans l'arbre, elles se trouvent sous un nœud de label `th` dont le père `tr` n'est pas le premier fils du nœud `table`, *i.e.* pas dans la première ligne. Les valeurs de la composante `Score` sont situées dans les autres cellules de la table. Dans l'arbre, leur père est le nœud `td`. `(Marseille,Marseille,-)`, `(Monaco,Marseille,1)` et `(Bordeaux,Monaco,6)` sont des exemples de triplets à extraire. Les valeurs de deux premières composantes sont, respectivement, en tête d'une ligne et d'une colonne et la valeur de la troisième composante est contenue dans la cellule à l'intersection de cette ligne et de cette colonne.

Ce type d'organisation possède des propriétés des deux cas précédents. Tout d'abord, comme on le constate sur la figure 3.11, les n-uplets sont entrelacés, à la fois dans la vue arborescente et séquentielle. Ensuite il y a une double factorisation. Les valeurs situées en tête d'une ligne, respectivement d'une colonne, sont factorisées avec les valeurs de la même ligne, respectivement colonne. Enfin comme dans le cas des tables tournées, les valeurs d'une même colonne se retrouvent à la même position dans l'arbre. Par

[5] http://www.imf.org



	Marseille	**Monaco**	**Bordeaux**
Marseille	-	1	2
Monaco	4	-	6
Bordeaux	2	2	-

FIG. 3.11 – Table croisée. Les n-uplets sont entralecés à la fois dans l'arbre et la vue feuillage.

exemple, dans le triplet `(Bordeaux,Monaco,6)` les valeurs `Bordeaux` et `6` ont un père qui est en quatrième position dans la séquence de ses frères. Cela traduit le fait que ces deux valeurs sont toutes les deux dans la quatrième colonne de la table. Pour associer correctement ces valeurs il est nécessaire de compter des positions dans l'arbre. Le langage d'arbres associé n'est donc pas régulier.

3.3.6 Caractéristiques

Dans une table croisée, il faut remarquer qu'une seule des deux valeurs factorisées n'est pas suffisante pour identifier un triplet. Par exemple avec le triplet `(Bordeaux,Monaco,6)` de la table de la figure 3.11, la feuille de l'arbre contenant la valeur `Bordeaux` ne désigne pas de manière unique ce tuple, car cette feuille appartient aussi à d'autres triplets, comme `(Bordeaux,Marseille,2)`. Par contre la feuille contenant la valeur `6` détermine clairement le triplet `(Bordeaux,Monaco,6)` : elle n'apparaît dans aucun autre triplet de la table. Par analogie avec le domaine des bases de données, on appelle la composante correspondant à une telle valeur une *composante clé*, car elle agit comme un identifiant pour les n-uplets. Dans le cas d'une table, tournée ou pas, et des listes toutes les composantes sont des clés. Cependant une composante factorisée ne peut être une composante clé, comme on l'a vu précédemment. Néanmoins, la combinaison de plusieurs composantes permet de former une clé. Dans l'exemple précédant, les deux feuilles `Bordeaux` et `Monaco`, respectivement en tête de colonne et de ligne, caractérisent de manière unique le triplet `(Bordeaux,Monaco,6)`.

Quelque soit l'organisation, l'extraction des n-uplets est rendue difficile lorsque certaines valeurs d'un tuple sont absentes. Au sein d'une table l'effet d'une valeur manquante est minime et se traduit généralement par une cellule vide. La structure locale de l'arbre n'est pas perturbée. Ce n'est pas le cas avec une liste, plate ou imbriquée, dans laquelle l'absence d'un élément modifie la structure de l'arbre.

Des variations dans l'organisation des données sont fréquentes. Elles consistent par exemple à fusionner certaines feuilles de l'arbre ou à mélanger plusieurs des cinq cas étudiés. Par exemple, la figure 3.12 présente une table du site Web BEA www.bea.gov. La relation n-aire cible est (`Country`, `Year`, `Exports`, `Balance`) et les quadruplets à extraire sont (`France`, `1986`, `10.130`, `7.119`), (`France`, `1987`, `11.701`, `7.947`), (`Germany`, `1986`, `14.738` ,`10.461`) et (`Germany`, `1987`, `17.184`, `11.525`). Les composantes `Year`, `Exports`, et `Balance` sont stockées dans une table tournée, tandis que le nom du pays, composante `Country`, est factorisé et placé dans la légende de la table correspondante. La figure 3.13 représente l'arbre `HTML` simplifié de cette page.

3.3.7 Bilan

Dans les deux premières organisations, les n-uplets sont stockés consécutivement, alors que pour les trois dernières ils sont entrelacés à la fois dans la vue séquentielle et arborescente.

La vue séquentielle des documents est inadaptée aux cas des tables tournées et croisées. En effet dans ces cas là, il est nécessaire de compter des positions dans l'arbre pour extraire les n-uplets, car les valeurs des composantes d'un même tuple apparaissent

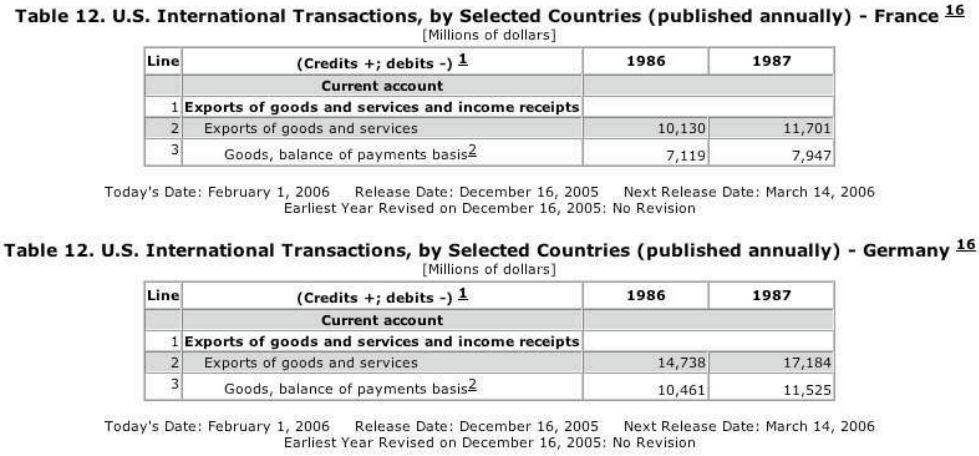


Table 12. U.S. International Transactions, by Selected Countries (published annually) - France [16]
[Millions of dollars]

Line	(Credits +; debits -) [1]	1986	1987
	Current account		
1	**Exports of goods and services and income receipts**		
2	Exports of goods and services	10,130	11,701
3	Goods, balance of payments basis[2]	7,119	7,947

Today's Date: February 1, 2006 Release Date: December 16, 2005 Next Release Date: March 14, 2006
Earliest Year Revised on December 16, 2005: No Revision

Table 12. U.S. International Transactions, by Selected Countries (published annually) - Germany [16]
[Millions of dollars]

Line	(Credits +; debits -) [1]	1986	1987
	Current account		
1	**Exports of goods and services and income receipts**		
2	Exports of goods and services	14,738	17,184
3	Goods, balance of payments basis[2]	10,461	11,525

Today's Date: February 1, 2006 Release Date: December 16, 2005 Next Release Date: March 14, 2006
Earliest Year Revised on December 16, 2005: No Revision

FIG. 3.12 – Une page du site Web BEA

à la même position dans différents sous-arbres. La vue arborescente est suffisamment expressive pour traiter l'extraction n-aire dans les cinq cas d'organisations des n-uplets que nous avons examinés. C'est sur elle qu'est basé notre système d'induction d'extracteurs (décrit dans le chapitre 4).

3.4 Expressivité des systèmes majeurs d'induction d'extracteurs

L'objectif de cette section est d'étudier l'expressivité de cinq systèmes d'extraction d'information. Le fonctionnement des extracteurs de ces cinq systèmes est présenté succinctement dans cette section. Pour une description plus détaillée, à la fois des algorithmes d'extraction et d'induction de ces systèmes, nous renvoyons le lecteur à l'annexe A. L'expressivité de ces systèmes est ici observée sous l'angle des organisations arborescentes mises en évidence dans la section 3.3, à savoir :

- les tables ;
- les listes ;
- les tables tournées ;
- les organisations avec des valeurs factorisées ;
- et les tables croisées.

Cette étude est abordée par une analyse des algorithmes d'extraction et de la représentation des exemples.

En apprentissage automatique, il est bien connu que la comparaison de différentes méthodes et algorithmes est délicate. Plusieurs articles ou ouvrages proposent des méthodes et des tests statistiques pour déterminer, parmi les approches mises en compétition, laquelle est significativement la plus efficace [25].

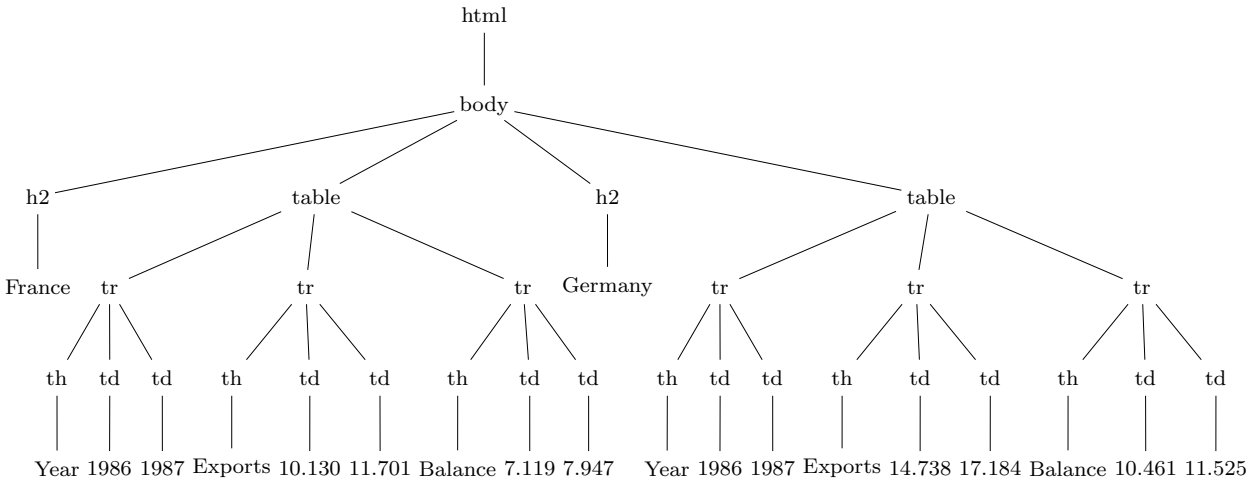


FIG. 3.13 – Arbre `HTML` simplifié d'une page du site Web BEA (voir la figure 3.12). Les noms des pays sont des valeurs factorisées, alors que les autres composantes sont stockées dans une table tournée

Hélas, on ne peut que déplorer que le domaine de l'extraction d'information ne fasse pas l'usage de ces techniques. À part SQUIRREL$_n$ les autres systèmes supervisés ne sont pas distribués et donc indisponibles. De plus certains d'entre eux font l'objet d'une commercialisation, comme STALKER. L'indisponibilité des systèmes rend leur comparaison expérimentale impossible.

À cela s'ajoute aussi l'impossibilité de comparer les résultats expérimentaux des différents articles. En effet à la fois les conditions expérimentales et les méthodes d'évaluation des résultats de l'extraction sont rarement décrites, ou alors trop peu, de façon trop évasive. Faisant leur *mea culpa*, certains de ces auteurs dénoncent ces pratiques dans [10]. Cet article accuse ce manque de rigueur en soulignant l'évidente impossibilité de comparer des expériences réalisées dans des conditions différentes et propose un vocabulaire et une méthode pour spécifier sans ambiguïté le protocole d'évaluation d'un système d'extraction. Il envisage même la création d'un serveur d'évaluation mis à la disposition de toute la communauté par le biais d'internet. Cet objectif n'est pas atteint à ce jour.

Ainsi, la seule manière restant à notre disposition pour appréhender l'expressivité des systèmes supervisés, étudiés dans la section A, est l'étude et l'analyse de leurs algorithmes, tels qu'ils sont décrits dans la littérature. Néanmoins, comme le remarque également [10], mêmes les algorithmes peuvent être décrits partiellement ou de façon obscure.

3.4.1 WIEN

Les extracteurs n-aires induits par WIEN [51] travaillent sur la représentation séquentielle des documents semi-structurés. Ils sont définis par n paires de délimiteurs $((l_1, r_1), \ldots, (l_n, r_n))$ et par une fonction d'extraction. Chaque paire de délimiteurs permet de caractériser une composante d'un n-uplet, l_i étant la valeur se trouvant à gauche de la valeur à

extraire (délimiteur gauche) et r_i juste à sa droite (délimiteur droit) dans la séquence du document.

Les extracteurs proposés par WIEN s'appliquent uniquement aux documents organisés sous forme *tabulaire* [51]. Dans une telle organisation les n-uplets sont stockés séquentiellement l'un après l'autre et leurs composantes sont dans un ordre unique et fixe.

[51] définit plus formellement une organisation tabulaire des n-uplets. La valeur d'une composante est représentée par le couple b, e de ses indices de début et de fin dans la séquence de caractères du document dont elle est issue. Ainsi l'ensemble des n-uplets d'un document est représenté par l'ensemble

$$\{((b_{1,1}, e_{1,1}), (b_{2,1}, e_{2,1}), \ldots, (b_{n,1}, e_{n,1})), \ldots, ((b_{1,k}, e_{1,k}), (b_{2,k}, e_{2,k}), \ldots, (b_{n,k}, e_{n,k}))\}$$

où k est le nombre de n-uplets à extraire. Ces n-uplets sont organisés de manière tabulaire si les contraintes suivantes sont vérifiées :

– pour chaque valeur, son indice de début est plus petit que son indice de fin

$$\forall i \in \{1, \ldots, n\} \quad \forall j \in \{1, \ldots, k\} \quad b_{i,j} \leq e_{i,j} \tag{3.1}$$

– les valeurs des n-uplets ne se chevauchent pas

$$\forall i \in \{1, \ldots, n\} \quad \forall j \in \{1, \ldots, k\} \quad \neg\exists\, i', j' \mid (b_{i,j} \leq b_{i',j'} \leq e_{i,j}) \vee (b_{i,j} \leq e_{i',j'} \leq e_{i,j}) \tag{3.2}$$

– toutes les composantes d'un n-uplet précèdent celles du n-uplet suivant

$$\forall j < k \quad e_{n,j} < b_{1,j+1} \tag{3.3}$$

– les composantes sont toujours dans le même ordre

$$\forall i < n \quad \forall j \in \{1, \ldots, k\} \quad e_{i,j} < b_{i+1,j} \tag{3.4}$$

D'après 3.1, 3.2 et 3.4 on a :

$$\forall j \in \{1, \ldots, k\} \quad b_{1,j} \leq e_{1,j} < b_{2,j} \leq e_{2,j} < \ldots < b_{n,j} \leq e_{n,j} \tag{3.5}$$

Et d'après 3.3 et 3.5, on obtient :

$$\begin{array}{rc} & b_{1,1} \leq e_{1,1} < b_{2,1} \leq e_{2,1} < \ldots < b_{n,1} \leq e_{n,1} \\ < & b_{1,2} \leq e_{1,2} < b_{2,2} \leq e_{2,2} < \ldots < b_{n,2} \leq e_{n,2} \\ < & \ldots \\ < & b_{1,k} \leq e_{1,k} < b_{2,2} \leq e_{2,k} < \ldots < b_{n,k} \leq e_{n,k} \end{array} \tag{3.6}$$

Parmi les cinq organisations de base identifiées à la section 3.3, seules deux d'entre elles sont tabulaires : les tables et les listes. En effet avec ces deux organisations les n-uplets sont consécutifs, ne se chevauchent pas et leurs composantes suivent toujours le même ordre.

```
<html> <body> <table>
<tr> <th>Club</th> <th>Score</th> </tr>
<tr> <td> PSG </td> <td> 17 </td> </tr>
          ↑b1,1 ↑e1,1          ↑b2,1↑e2,1
<tr> <td> OM  </td> <td> 31 </td> </tr>
          ↑b1,2 ↑e1,2          ↑b2,2↑e2,2
</table> </body> </html>
```

FIG. 3.14 – Organisation tabulaire : une table d'un document `HTML`

```
<html> <body> <table>
<tr> <th>Club</th> <td>  PSG </td> <td> OM </td> </tr>
                        ↑b1,1 ↑e1,1          ↑b1,2↑e1,2
<tr> <th>Score</th> <td> 17  </td> <td> 31 </td> </tr>
                        ↑b2,1 ↑e2,1          ↑b2,2↑e2,2
</table> </body> </html>
```

FIG. 3.15 – Organisation non tabulaire : une table tournée contenant les même données que la table de la figure 3.14

La figure 3.14 montre le code source d'un document `HTML` dans lequel les n-uplets sont stockés dans une table. La relation binaire cible est `(Club,Score)` et les couples à extraire `(PSG,17)` et `(OM,31)` sont représentés respectivement par $((b_{1,1}, e_{1,1}), (b_{2,1}, e_{2,1}))$ et $((b_{1,2}, e_{1,2}), (b_{2,2}, e_{2,2}))$. La figure 3.14 illustre bien le fait qu'on a :

$$b_{1,1} \leq e_{1,1} < b_{2,1} \leq e_{2,1} < b_{1,2} \leq e_{1,2} < b_{2,2} \leq e_{2,2}$$

et la contrainte 3.6 est vérifiée. Elle le serait également avec une liste.

Par contre lorsque ces même données sont rangées dans la table tournée de la figure 3.15, la contrainte 3.6 est violée. En effet, on peut constater sur la figure 3.15 qu'on a cette fois :

$$b_{1,1} \leq e_{1,1} < b_{1,2} \leq e_{1,2} < b_{2,1} \leq e_{2,1} < b_{2,2} \leq e_{2,2}$$

Ainsi la table tournée de la figure 3.15 n'est pas tabulaire et ce résultat se généralise aisément à toute table tournée.

L'organisation de la figure 3.16 n'est pas tabulaire non plus. La relation cible est maintenant `(Club,Score,Season)` et les triplets à extraire sont `(PSG,17,2002)` et `(OM,31,2002)`, représentés respectivement par $((b_{1,1}, e_{1,1}), (b_{2,1}, e_{2,1}), (b_{3,1}, e_{3,1}))$ et $((b_{1,2}, e_{1,2}), (b_{2,2}, e_{2,2}), (b_{3,2}, e_{3,2}))$. La valeur de la composante `Season` est partagée par les deux triplets. Cette composante est factorisée et c'est pourquoi $b_{3,1} = b_{3,2}$ et $e_{3,1} = e_{3,2}$. À nouveau la contrainte 3.6 n'est pas respectée car on a :

$$b_{3,1} = b_{3,2} \leq e_{3,1} = e_{3,2} < b_{1,1} \leq e_{1,1} < b_{2,1} \leq e_{2,1} < b_{1,2} \leq e_{1,2} < b_{2,2} \leq e_{2,2}$$

```
<html> <body> <ul> <li> Season
<b>       2002       </b> </li> </ul> <table>
   ↑b3,1=b3,2      ↑e3,1=e3,2
<tr> <th>Club</th> <th>Score</th> </tr>
<tr> <td> PSG </td> <td> 17 </td> </tr>
          ↑b1,1 ↑e1,1          ↑b2,1↑e2,1
<tr> <td> OM  </td> <td> 31 </td> </tr>
          ↑b1,2 ↑e1,2          ↑b2,2↑e2,2
</table> </body> </html>
```

FIG. 3.16 – Organisation non tabulaire : l'organisation du document `HTML` de la figure 3.8 (à la page 64) avec des valeurs factorisées http:www.fr

L'ordre dans lequel on considère les composantes de la relation ne remet pas en cause cette observation, car dans le document la valeur de la composante `Season` du deuxième triplet se trouve avant, dans le parcours du document, les autres composantes du premier triplet. Les deux triplets se chevauchent, et donc l'organisation n'est pas tabulaire.

Enfin comme une table croisée n'est pas non plus une organisation tabulaire. En effet, dans une telle table des valeurs sont factorisées sur les lignes et les colonnes. De plus une partie des données est stockée comme dans une table tournée.

Ainsi WIEN ne peut traiter l'extraction de données depuis les tables tournées, les organisations avec des valeurs factorisées et les tables croisées. Cependant il ne faut oublier que WIEN a été élaboré en 1997, date à laquelle les standards du Web n'étaient pas ceux d'aujourd'hui. La façon de concevoir les documents `HTML` n'était pas aussi évoluée. Dans ce sens, l'approche adoptée par WIEN était adaptée à la structure des documents arborescents de cette époque.

3.4.2 SOFTMEALY

Dans SOFT MEALY, un extracteur est composé de deux transducteurs : l'un pour trouver la zone du document contenant les données, l'autre pour extraire les n-uplets. Le transducteur de tuples extrait un tuple à la fois. Il est appliqué tant que la zone des données n'est pas parcourue intégralement. Un tel procédé d'extraction suppose que les n-uplets ne se chevauchent pas et qu'ils sont consécutifs dans le parcours du document. Il permet de traiter les tables et les listes, mais pas les tables tournées, croisées et les organisations avec factorisation. En effet nous avons déjà vu que dans ces trois derniers cas, les n-uplets sont entrelacés à la fois dans la structure arborescente du document, mais aussi dans la vue textuelle utilisée par SOFT MEALY.

3.4.3 STALKER

Un extracteur de STALKER est constitué d'une description de l'organisation des données dans le document sous la forme d'un arbre, dit arbre EC, et d'un ensemble

de règles d'extractions, à raison d'une règle par nœud de l'arbre. Les feuilles de l'arbre EC correspondent aux composantes à extraire. Le formalisme de l'arbre EC permet de décrire des organisations constituées de listes, éventuellement imbriquées, de valeurs. Bien que cela permette de traiter des composantes multivaluées, ce formalisme est surtout destiné aux organisations dans lesquelles les n-uplets sont stockés séquentiellement. Il est donc limité aux tables et aux listes.

De plus dans [65] il est indiqué que l'extraction d'une composante est réalisée indépendamment des autres. Il n'est donc pas possible d'extraire des données depuis une table tournée ou croisée, dans lesquelles les composantes d'un même n-uplet se retrouvent à la même position dans l'arbre. Les règles d'extraction de STALKER repèrent des délimiteurs. En aucun cas elles ne permettent de compter et de mémoriser des positions dans le document.

3.4.4 LIPX

Dans LIPX un extracteur est une clause dont les prédicats décrivent des contraintes sur les vues séquentielle et arborescente et dont les variables correspondent aux composantes à extraire. LIPX tolère les valeurs manquantes et ne fait pas d'hypothèses sur l'ordre des composantes dans le document.

A priori le langage de description des spans est assez expressif. Il permet notamment de désigner des positions identiques dans un arbre. Ainsi il doit pouvoir décrire les tables tournée et croisée. Cependant, ce qui n'est pas clair c'est la capacité de LIPX a décrire des propriétés, arborescentes ou textuelles, qui ne se limitent pas à un span à extraire ou au nœud qui lui est associé. Il est difficile d'envisager l'étendue du contexte que le langage de représentation de LIPX peut décrire.

3.4.5 SQUIRREL$_n$

Les extracteurs de SQUIRREL$_n$ sont des transducteurs de sélection de n-uplets de nœuds, dont l'expressivité est celle de la classe des requêtes n-aires définissables en logique du second ordre monadique. Nous avons vu (à la section A.5) qu'une requête n-aire est équivalente à un automate d'arbre sur $\Sigma \times Bool^n$. Un tel automate permet de traiter uniquement les langages d'arbres réguliers. Ainsi SQUIRREL$_n$ ne peut assurer l'extraction de n-uplets stockés dans une table tournée et dans une table croisée. Aucun langage régulier ne permet d'engendrer ces deux structures. Les expériences de [56] confirment ce résultat.

Les résultats expérimentaux de [56] montrent aussi que l'extraction échoue si la structure arborescente du document n'est pas suffisamment riche. C'est par exemple le cas des listes de n-uplets construites sans l'utilisation des balises `HTML ul` et `li` et dans lesquelles seule la prise en compte du contenu textuel au voisinage des données permet leur extraction.

3.5 Conclusion

Ce chapitre précisé quelle tâche d'extraction d'information n-aire nous considérons : l'extraction de n-uplets de feuilles textes dans les documents `HTML` représentés comme des arbres. Ensuite il présente notre étude des différentes manières de stocker des n-uplets dans un document arborescent. Cinq cas de base se dégagent :

- les tables ;
- les listes ;
- les tables tournées ;
- les organisations avec factorisation ;
- les tables croisées.

Enfin les capacités de différents systèmes d'extraction d'information sont étudiées en les confrontant aux cinq organisations de base. Les systèmes supervisés présentés dans la section A sont observés à nouveau du point de vue des organisations. Ils sont tous spécifiques à un type de document et incapables de traiter toutes les organisations de base.

Chapitre 4

Induction d'extracteurs n-aire

4.1 Introduction

Ce chapitre présente notre approche de l'extraction d'information n-aire. La difficulté majeure du cas n-aire est que le nombre de n-uplets candidats à l'extraction est exponentiel en n. De plus il est à la fois nécessaire d'extraire correctement les composantes et de construire correctement les tuples.

Pour répondre à ces deux difficultés, notre approche est fondée sur deux principes. Le premier est *l'extraction incrémentale* : l'extraction des n-uplets est réalisée itérativement en considérant d'abord les singletons, puis les couples, les triplets,... jusqu'à l'obtention des n-uplets. Cette approche incrémentale est justifiée pour des raisons de complexité. Ainsi le nombre de tuples candidats à l'extraction n'est plus exponentiel en n, mais borné à chacune des n étapes d'extraction. Elle se justifie également par l'idée que l'extraction d'un tuple de longueur i est facilitée par la connaissance de ses $i-1$ premières composantes. Ce qui nous conduit au second principe de base de notre approche : la notion d'*enrichissement de la représentation*. À chaque étape i de l'algorithme d'extraction, les représentations en attribut valeur des tuples incomplets de longueur i intègrent des informations provenant des tuples extraits de longueur $i-1$.

Nos algorithmes d'extraction et d'induction s'inspirent des schémas algorithmiques génériques présentés dans les sections 2.4.2 et 2.4.3. Les aspects représentation du document, nature et codage des exemples et algorithme d'apprentissage sont clairement identifiés et séparés. Les documents sont représentés comme des arbres et les exemples sont des tuples de feuilles textes. Leur codage en attribut valeur fait appel principalement à des propriétés locales de la structure arborescente des documents. Le codage que nous proposons permet de traiter les cinq cas de base d'organisation des données identifiés dans le chapitre précédant (voir le chapitre 3). Comme algorithme d'apprentissage supervisé, nous utilisons les méthodes présentées dans la section 2.2.3. La validité de notre approche est démontrée expérimentalement sur des jeux de données classiques en extraction d'information, pour lesquels nous obtenons des performances supérieures ou égales aux systèmes classiques décrits dans la section A. Mais ces corpus sont trop simples et les données qu'ils contiennent sont organisées seulement en liste et en table. C'est pourquoi l'évaluation s'effectue aussi sur des corpus produits automa-

tiquement selon les organisations de base identifiées dans le chapitre précédent. et des sites représentatifs du Web actuel et qui sont des illustrations concrètes des cas de base d'organisations des données, notamment les cas les plus durs comme les tables tournées et croisées. Notre approche permet de traiter ces cas, alors que les approches classiques échouent.

4.2 Cadre de travail

Précisons notre cadre de travail. Nous traitons le problème d'extraction de n-uplets, instances d'une relation n-aire, telle que la valeur de chaque composante est exactement une feuille texte de l'arbre correspondant au document `HTML` ou `XML`.

Avec notre approche incrémentale, l'extraction a lieu selon un certain ordre sur les composantes ; cet ordre étant celui utilisé au cours du processus d'induction. Il est évident que certains ordres sur des composantes sont plus favorables que d'autres. Cependant la connaissance de l'ordre idéal n'a d'influence que sur les performances de l'extracteur induit, elle n'est pas nécessaire pour décrire le fonctionnement des algorithmes. Ainsi on considérera que l'ordre des composantes de la relation n-aire cible est fixé. Nous verrons même que pour certaines organisations des données, la qualité des résultats n'est pas dépendante de cet ordre. En pratique, la détermination de l'ordre optimal des composantes ne s'est pas avéré être un problème majeur. Enfin on peut aisément modifier l'algorithme d'induction afin qu'il détermine lui-même cet ordre.

Dans ce chapitre, nous considérons uniquement l'induction d'extracteurs d'après des documents complètement étiquetés. L'annotation complète d'un document consiste à exprimer l'ensemble des n-uplets à en extraire explicitement par énumération. Un tuple à extraire est aussi désigné par l'expression exemple positif, tandis qu'un tuple à ne pas extraire est un exemple négatif. Pour un document complètement annoté, l'ensemble des exemples négatifs est implicite : tout n-uplet qui n'est pas positif est nécessairement négatif. Le nombre d'exemples négatifs étant exponentiel en n, leur énumération explicite n'est pas possible.

4.3 Aperçu de l'approche

Avant de décrire précisément les algorithmes d'extraction et d'induction de notre approche, nous allons illustré le fonctionnement général des extracteurs n-aire que nous manipulons en décrivant leur procédé d'extraction incrémentale. C'est sur ce mécanisme itératif d'extraction qu'est bâti l'ensemble de notre système, qu'il s'agisse de la représentation des tuples, de l'algorithme d'extraction ou encore de l'algorithme d'induction.

Dans notre approche, un extracteur n-aire extrait les n-uplets d'un document de manière incrémentale. Ce processus débute par l'extraction de singletons à partir desquels seront extraits des couples qui eux même permettront l'extraction de triplets *etc* jusqu'à l'obtention des n-uplets. Ainsi le procédé d'extraction est composé de n étapes et chacune d'entre elles consiste à extraire des tuples de taille i à partir des tuples de taille $i-1$ extraits lors de l'étape précédente.

Le problème de l'extraction des tuples de taille i est transformé en un problème de classification binaire de tuples de taille i selon les deux classes "à extraire" et "à ne pas extraire". Ainsi un extracteur n-aire est une séquence de n classificateurs binaires. Ces n tâches de classification sont enchaînées selon une boucle variant de 1 à n sur la taille des tuples.

Le schéma de la figure4.1 illustre le fonctionnement d'un extracteur n-aire sur la page Web se trouvant en haut à gauche de la figure. Il s'agit ici d'extraire les couples `(Club,Score)`. On suppose qu'on dispose d'un extracteur $w = (c_1, c_2)$ constitué de deux classificateurs c_1 et c_2 pour réaliser cette tâche.

Un traitement préliminaire consiste à calculer la représentation du document d'entrée. Dans notre approche, les documents semi-structurés sont représentés comme des arbres d'arité arbitraire. À partir de là débute la boucle sur les composantes évoquée auparavant. Dans cet exemple, cette boucle comporte deux étapes :

- l'extraction des singletons ;
- puis celle des couples.

Chacune des deux phases d'extraction est traitée comme une tâche de classification constituée des étapes suivantes :

- constitution de l'ensemble des tuples de taille i candidats à l'extraction d'après les tuples de taille $i - 1$ extraits ;
- codage des tuples ;
- application du classificateur c_i dont la sortie est les tuples de taille i extraits.

Lors de la première étape, l'ensemble des feuilles de l'arbre du document constitue l'ensemble des singletons candidats présentés au classificateur c_1. En sortie de c_1, on s'intéresse uniquement aux singletons classés comme étant à extraire. C'est à partir d'eux qu'est construit l'ensemble des couples candidats à l'extraction de la seconde étape. Un couple est obtenu en complétant un singleton extrait par une feuille de l'arbre du document. Les couples ainsi construits sont fournis au classificateur c_2 et ceux classés comme à extraire sont le résultat de l'extraction.

Avant de décrire plus en détail l'approche que nous venons d'esquisser, il est nécessaire de préciser la représentation de documents et celles des tuples. Ensuite nous décrirons l'algorithme d'extraction et celui d'induction.

4.4 Représentation des documents

Nous avons choisi de représenter les documents `HTML` à l'aide de deux vues :

- la vue arborescente ;
- et la vue *feuillage* obtenue à partir de la précédente.

Elles sont illustrées par la figure 4.2. La vue arborescente est induite par l'imbrication des balises des documents `HTML` et `XML` qui incite à les voir naturellement comme des arbres. De plus l'étude des différentes organisations des n-uplets dans des documents semi-structurés (chapitre 3) a montré la pertinence de cette vue. La vue feuillage, quand à elle, s'intéresse uniquement au feuillage d'un arbre. Il s'agit de la séquence des feuilles de l'arbre [36, 37] obtenue par un parcours en profondeur et à gauche d'abord.

Précisons que dans le cas de la vue arborescente, les attributs des documents `HTML` et `XML` ne sont pas représentés dans cette vue. Ajoutons également que la représentation

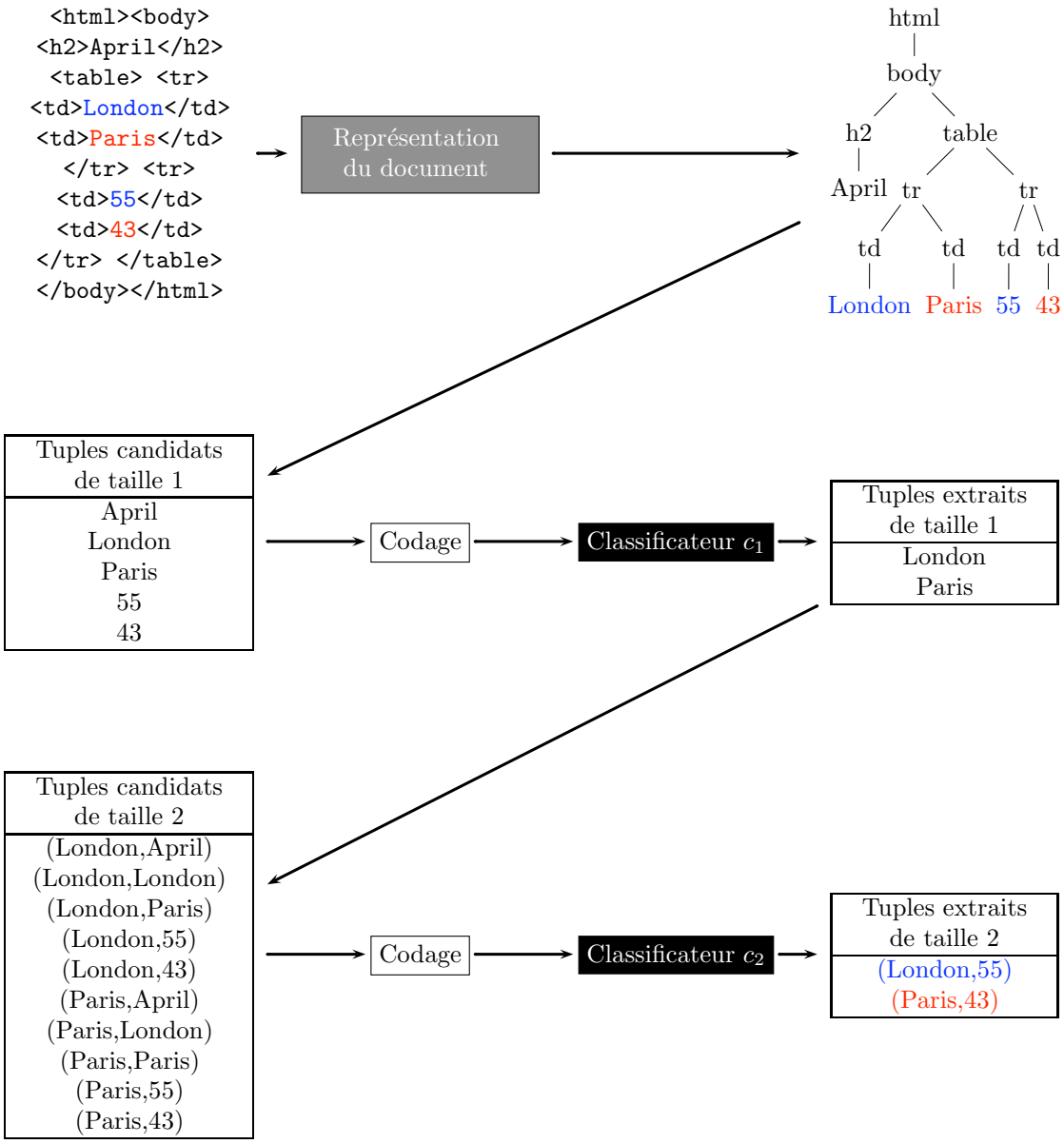


FIG. 4.1 – Fonctionnement schématique de l'algorithme d'extraction 4

choisie pour les arbres est celle des arbres d'arité non bornée. Un arbre d'arité non bornée, ou arité arbitraire, est un arbre dont le nombre de fils d'un nœud n'est pas borné. Par exemple, en `HTML`, la balise `ul` qui annonce une liste non-numérotée. Chaque élément de la liste est désigné par la balise `li`. Dans l'arbre `HTML` chaque nœud `li` a comme père le nœud `ul`, constructeur de la liste. Une telle liste peut avoir un nombre quelconque d'éléments : le nombre de fils d'un nœud `ul` n'est pas borné.

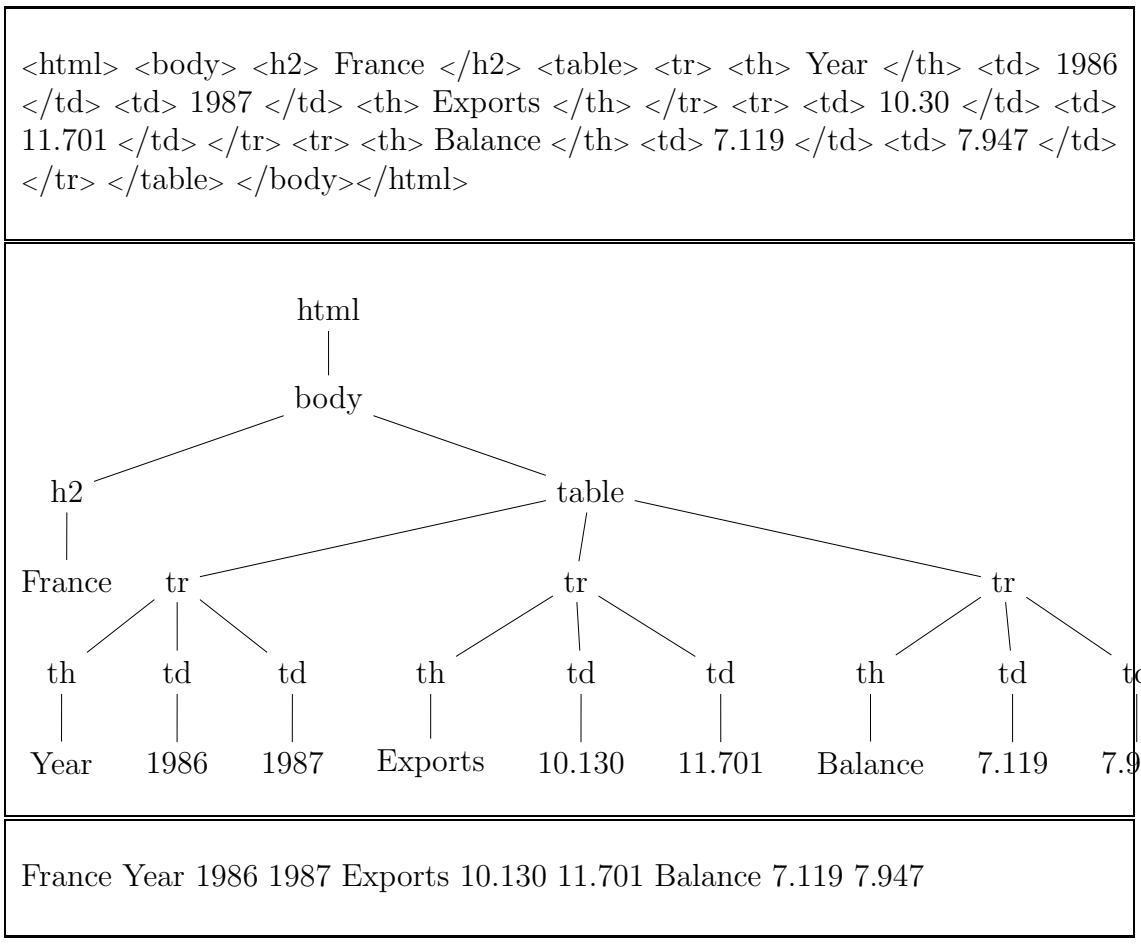


FIG. 4.2 – Trois vues d'un document arborescent : séquentielle (en haut), arborescente (au milieu), feuillage (en bas)

4.5 Représentation des n-uplets

Les documents sont représentés comme des arbres et les exemples sont des n-uplets de feuilles textes. La représentation des n-uplets est un codage attribut valeur utilisant la vue arborescente et la vue feuillage des documents (voir la section 4.4 à la page 79).

Ce codage est facilement extensible pour intégrer des connaissances du domaine, ou d'autres vues sur les données. Pour ce faire, il suffit d'ajouter les attributs adéquats.

4.5.1 Codage d'un nœud

Tout nœud de l'arbre est codé par des attributs décrivant le nœud lui-même, mais également son contexte dans l'arbre.

Un nœud n est codé par les attributs suivants :

- le label de n (une valeur spéciale est utilisée s'il s'agit d'une feuille texte) ;
- sa position dans la séquence des fils de son père ;
- sa profondeur dans l'arbre ;
- sa hauteur dans l'arbre ;
- le nombre de ses fils ;
- la taille (en nombre nœuds) du sous arbre enraciné en n ;
- le label de son frère gauche (une valeur spéciale est utilisée si n n'a pas de frère gauche) ;
- le label de son frère droit (une valeur spéciale est utilisée si n n'a pas de frère gauche) ;

Dans le cas des documents `HTML`, on considère aussi la valeur de l'attribut `HTML` *class* (une valeur spéciale est utilisée si n ne possède pas l'attribut *class*) qui apporte une certaine sémantique aux nœuds qui le portent. Ainsi un nœud est décrit par 9 attributs illustrés par la figure 4.3.

4.5.2 Codage d'une feuille

Une feuille est codée comme tout nœud de l'arbre à l'aide des 9 attributs précédents (voir la section 4.5.1), ainsi qu'avec les attributs suivants :

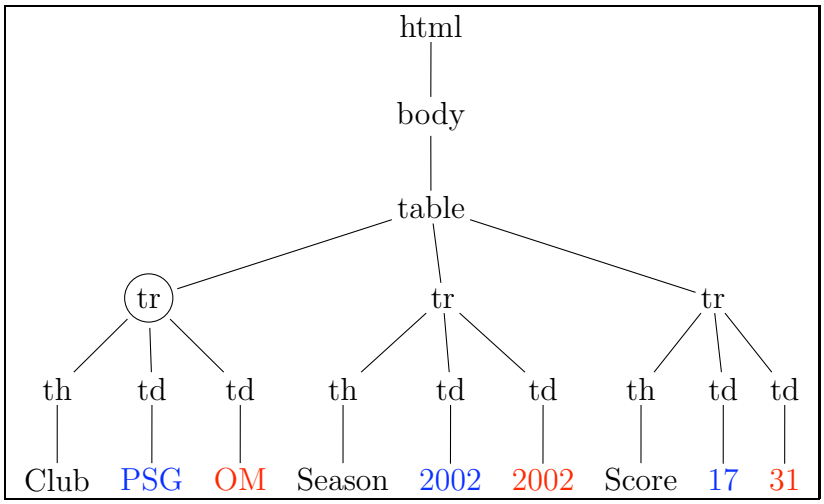
- le codage des cinq ancêtres de la feuille, à l'aide du codage des nœuds de la section précédente (45 attributs au total) ;
- le contenu textuel de la feuille précédente dans la vue feuillage (sans tokenisation), si cette feuille n'existe pas, une valeur spéciale est utilisée ;
- le contenu textuel de la feuille suivante dans la vue feuillage.

Ainsi une feuille est représentée par 56 attributs qui mélangent à la fois les vues arborescentes et feuillage. La figure 4.4 illustre la représentation d'une feuille.

4.5.3 Codage d'une dépendance entre deux feuilles

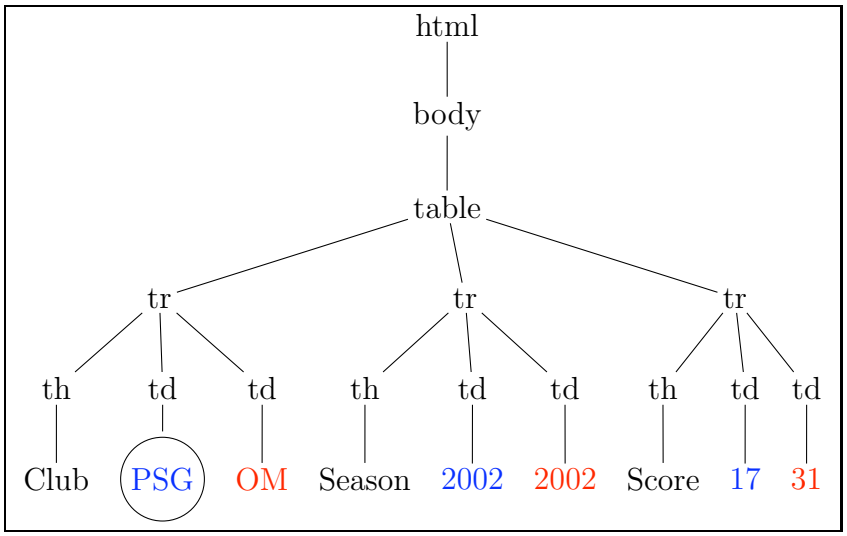
Dans notre approche incrémentale, les tuples de taille i sont extraits avec la connaissance des tuples de taille $i-1$. Pour représenter la dépendance entre l_i, la $i^{\text{ième}}$ composante ajoutée à un tuple $(l_1, \ldots, l_{i-1})$ de taille $i-1$, et l'une des $i-1$ premières composantes nous utilisons des attributs qui modélisent la relation entre deux feuilles.

Considérons deux feuilles p et m. Soit a le nœud qui est le plus petit ancêtre commun à p et m. Ainsi a est ancêtre de p et de m et il n'existe pas de nœud o tel que o soit ancêtre de p et m et a est ancêtre de o. Le codage de la relation entre p et m se fait grâce aux attributs suivants :



Label	tr
Position	1
Profondeur	3
Hauteur	2
Nombre de fils	3
Taille	7
Label du frère gauche	RIEN
Label du frère droit	tr
Valeur de *class*	RIEN

FIG. 4.3 – Codage du nœud tr encerclé



Label	TEXTE
Position	1
Profondeur	5
Hauteur	0
Nombre de fils	0
Taille	1
Label du frère gauche	TEXTE
Label du frère droit	TEXTE
Valeur de *class*	RIEN
Valeur de la feuille précédente	Club
Valeur de la feuille suivante	OM
Label de l'ancêtre 1	td
Position de l'ancêtre 1	2
Profondeur de l'ancêtre 1	4
Hauteur de l'ancêtre 1	1
Nombre de fils de l'ancêtre 1	1
Taille de l'ancêtre 1	2
Label du frère gauche de l'ancêtre 1	td
Label du frère droit de l'ancêtre 1	td
Valeur de *class* de l'ancêtre 1	RIEN
Label de l'ancêtre 2	tr
Position de l'ancêtre 2	1
Profondeur de l'ancêtre 2	3
Hauteur de l'ancêtre 2	2
Nombre de fils de l'ancêtre 2	3
Taille de l'ancêtre 2	7
Label du frère gauche de l'ancêtre 2	RIEN
Label du frère droit de l'ancêtre 2	tr
Valeur de *class* de l'ancêtre 2	RIEN

Label de l'ancêtre 3	table
Position de l'ancêtre 3	1
Profondeur de l'ancêtre 3	2
Hauteur de l'ancêtre 3	3
Nombre de fils de l'ancêtre 3	3
Taille de l'ancêtre 3	22
Label du frère gauche de l'ancêtre 3	RIEN
Label du frère droit de l'ancêtre 3	RIEN
Valeur de *class* de l'ancêtre 3	RIEN
Label de l'ancêtre 4	body
Position de l'ancêtre 4	1
Profondeur de l'ancêtre 4	1
Hauteur de l'ancêtre 4	4
Nombre de fils de l'ancêtre 4	1
Taille de l'ancêtre 4	23
Label du frère gauche de l'ancêtre 4	RIEN
Label du frère droit de l'ancêtre 4	RIEN
Valeur de *class* de l'ancêtre 4	RIEN
Label de l'ancêtre 5	html
Position de l'ancêtre 5	RIEN
Profondeur de l'ancêtre 5	0
Hauteur de l'ancêtre 5	5
Nombre de fils de l'ancêtre 5	1
Taille de l'ancêtre 5	24
Label du frère gauche de l'ancêtre 5	RIEN
Label du frère droit de l'ancêtre 5	RIEN
Valeur de *class* de l'ancêtre 5	RIEN

FIG. 4.4 – Codage de la feuille PSG encerclée

- représentation de a selon le codage d'un nœud (voir la section 4.5.1) ;
- taille du chemin le plus court entre a et p (nombre d'arêtes rencontrées sur le chemin) dans l'arbre, ainsi que dans le codage premier fils, premier frère du même arbre ; on a ainsi deux attributs différents pour évaluer la distance entre a et p ;
- la même chose mais pour a et m ;
- la même chose mais pour p et m ;
- la distance, comptée en nombre de feuilles textes, dans la vue feuillage entre p et m, autrement dit la différence en valeur absolue entre la position de p dans la séquence des feuilles et celle de m ;
- la position relative de p par rapport à m, cet attribut prend soit la valeur *avant* soit la valeur *après* ;
- et enfin pour $1 \leq k \leq 5$, la différence, en valeur absolue, entre la position (par rapport à son père) du k-ième ancêtre de p et du k-ième ancêtre de m, ce groupe de cinq attributs permet de comparer une partie de la fin de la branche qui mène à m à celle qui mène à p.

Il y a 22 attributs impliqués dans la représentation de la relation entre deux feuilles, faisant intervenir à la fois la vue arborescente et la vue feuillage. Le groupe d'attributs portant sur la comparaison de branches est particulièrement utile lorsque les feuilles p et m sont au sein d'une table, dans laquelle les tuples sont stockés en colonne (voir la section 3.3.3). En effet, si pour p_j et m_j, j-ième ancêtre de p et m respectivement, la différence entre leur position par rapport à leur père est nulle, alors p et m sont dans la même colonne de la table.

4.5.4 Codage d'un tuple

Considérons un tuple $t_i = (l_1, \ldots, l_i)$ de taille i. Sa représentation en attribut valeur est constituée :

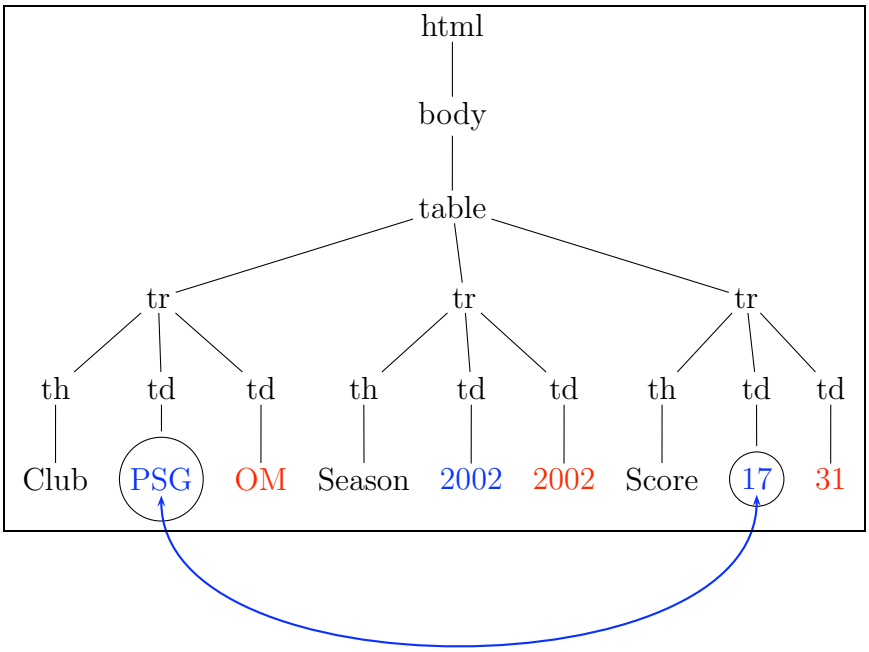
- du codage de l_i comme décrit dans la section 4.5.2 ;
- du codage de la relation entre l_i et le tuple incomplet $t_{i-1} = (l_1, \ldots, l_{i-1})$ à l'aide de dépendances.

La représentation en attribut valeur de t_i ne fait pas intervenir toutes ses composantes mais seulement celle de la dernière composante. Ce choix est justifié par la nature incrémentale du procédé d'extraction. Comme les tuples de taille i sont extraits à partir des tuples de taille $i-1$, seule la $i^{\text{ième}}$ composante d'un tuple de taille i nécessite d'être représentée, les autres composantes ayant été codées lors des étapes précédentes.

Les choix des dépendances pour représenter la relation entre l_i et le tuple incomplet t_{i-1} sont à nouveau déterminés par l'incrémentalité du procédé d'extraction : seules les dépendances liées à la composante i sont codées. Le champ des choix possibles est délimité par :

- d'un côté considérer la dépendance à la graine seulement, *i.e.* la dépendance entre l_i et l_1 ;
- de l'autre prendre en compte les dépendances à toutes les composantes, *i.e.* les dépendance entre l_i et l_j pour $1 \leq j \leq i-1$, contrairement au cas précédent le nombre d'attributs n'est pas fixe et augmente selon i.

Pour trancher entre ces deux extrêmes, nous considérons les contraintes suivantes :



Label du plus petit ancêtre commun	table
Position du plus petit ancêtre commun	1
Profondeur du plus petit ancêtre commun	2
Hauteur du plus petit ancêtre commun	3
Nombre de fils du plus petit ancêtre commun	3
Taille du plus petit ancêtre commun	22
Label du frère gauche du plus petit ancêtre commun	RIEN
Label du frère droit du plus petit ancêtre commun	RIEN
Valeur de *class* du plus petit ancêtre commun	RIEN
Distance entre PSG et le plus petit ancêtre commun	3
Distance dans l'arbre binarisé entre PSG et le plus petit ancêtre commun	4
Distance entre 17 et le plus petit ancêtre commun	3
Distance dans l'arbre binarisé entre 17 et le plus petit ancêtre commun	6
Distance entre PSG et 17	6
Distance dans l'arbre binarisé entre PSG et 17	8
Distance dans la vue feuillage entre PSG et 17	5
Position relative dans la vue feuillage de 17 par rapport à PSG	aprés
Différence de position entre les ancêtres 1 de 17 et de PSG	0
Différence de position entre les ancêtres 2 de 17 et de PSG	2
Différence de position entre les ancêtres 3 de 17 et de PSG	0
Différence de position entre les ancêtres 4 de 17 et de PSG	0
Différence de position entre les ancêtres 5 de 17 et de PSG	0

FIG. 4.5 – Codage de la dépendance de la feuille 17 à la feuille PSG

– intégrer des informations des $i-1$ premières composantes déjà extraites ;
– avoir un nombre d'attributs constant quel que soit i.

D'où notre choix des dépendances suivantes :

– dépendance à la graine, *i.e.* entre l_i et l_1 ;
– dépendance à l'avant dernière composante, *i.e.* entre l_i et l_{i-1}.

Un tuple de taille i est donc représenté par les 56 attributs du codage de la $i^{\text{ième}}$ feuille, plus 22 attributs pour chaque dépendance, soit 100 attributs au total.

4.5.5 Complexité du codage

Le codage d'un n-uplet est efficace.

Les attributs portant sur les noeuds et les feuilles sont calculés lors de la lecture du document (et de la construction de l'arbre correspondant). Leur coût de calcul est linéaire en la taille des arbres à traiter (en nombre de nœuds).

Le calcul du plus petit ancêtre commun entre deux feuilles, et le calcul des attributs des dépendances se font à la demande, uniquement lorsque cela est nécessaire, afin de ne pas faire de pré-calculs inutiles.

Le plus petit ancêtre commun entre deux nœuds d'un arbre est calculable en temps constant après un pré-traitement linéaire dans la taille de l'arbre correspondant [41]. Ce pré traitement peut s'effectuer lors du processus de lecture du document/création de l'arbre.

Les attributs qui interviennent dans le calcul des dépendances ont un coût linéaire. Pour deux feuilles p et m, une fois que leur plus petit ancêtre commun a est connu, le calcul de la distance entre p (ou m) et a est simplement la différence entre les profondeurs de a et de p (ou m) dans l'arbre. Ensuite, la distance entre p et m est la somme de la distance entre a et p, et celle entre a et m. Le calcul de ces distances sur le même arbre codé selon la représentation binaire premier frère/premier fils, s'effectue aisément sur la représentation classique (en arité non bornée) de l'arbre. Par exemple, pour le calcul de la distance entre p et a dans la représentation binaire de l'arbre, il suffit, pour chaque nœud o rencontré sur le chemin entre p et a (p y compris), de rajouter l'indice de o dans la séquence de ses frères (position de o par rapport à son père) moins un à la valeur de la distance entre p et a dans l'arbre classique.

4.6 Algorithme d'extraction

L'algorithme d'extraction 4 exposé ici est conçu selon un principe d'extraction incrémentale, des singletons aux n-uplets. À chaque étape, l'extraction est réalisée par un classificateur binaire de tuples de taille i, avec i variant de 1 à n. Après avoir décrit précisément l'algorithme et son fonctionnement, nous discutons de sa complexité puis nous exhibons ses propriétés.

4.6.1 Algorithme

L'algorithme 4 prend en entrée le document d à traiter ainsi que n classificateurs de tuples $c_1, \ldots, c_n$. Le procédé d'extraction est incrémental : l'extraction des n-uplets

est réalisée par l'extraction de tuples de longueur i avec le classificateur c_i, i variant de 1 à n. L'extraction des tuples de longueur i est ramenée à un problème de classification binaire qui consiste à décider si un i-uple est à extraire ou pas. Le processus incrémental est motivé par des raisons de complexité : dans un arbre, le nombre de tuples de feuilles de longueur n est exponentiel en n. Ainsi si l'arbre contient p feuilles, il y a p^n couples de feuilles candidats à l'extraction. Ce trop grand nombre de couples rend périlleuse une approche qui considère directement des n-uplets. Avec notre approche, à chaque étape i de l'algorithme le nombre de tuples de longueur i candidats à l'extraction est borné[1].

L'extraction est également fondée sur un processus d'enrichissement : les tuples de longueur i sont extraits avec la connaissance des tuples de longueur $i-1$. Ceci est motivé par le fait que l'extraction d'un tuple de longueur i est facilitée si ses $i-1$ premières composantes sont connues. De ce point de vue, le procédé d'extraction consiste, à chaque étape i, à rajouter une composante aux couples déjà extraits, et à ne conserver que les couples correctement étendus.

L'algorithme débute par l'extraction de la *graine*. Pour chaque n-uplet, il s'agit de la première composante extraite. C'est à partir des graines que les tuples de longueurs supérieures seront construits. L'ensemble des graines candidates S_1 est obtenu à la ligne 1. C'est l'ensemble tuples de longueur 1, *i.e.* des singletons, construits à partir de l'ensemble $L(d)$ des feuilles textes du document d. Le classificateur de singletons c_1 est appliqué sur chaque exemple de S_1, et dont la représentation est calculée à l'aide de la fonction rep_1 (ligne 2). rep_1 est la fonction de codage des feuilles. Elle utilise les attributs décrits à la section 4.5.2.

Le processus d'enrichissement commence à partir de la boucle de la ligne ligne 3. À l'étape i ($i \neq 1$), la construction des tuples t_i de longueur i (candidats à l'extraction) se fait à partir de l'ensemble S^+_{i-1} des tuples de longueur $i-1$ extraits lors de l'étape précédente. L'ensemble des tuples candidats S_i est obtenu par l'ajout d'une feuille de d à chaque tuple extrait de taille $i-1$ (ligne 4). Les tuples de taille i sont extraits ligne 5 grace au classificateur de tuples c_i. On obtient l'ensemble S^+_i. Le codage des tuples t_i est assuré par la fonction rep_i selon la représentation décrite à la section 4.5.4.

Il n'y a pas de filtrage des exemples : tous les t_i classés comme positif sont extraits. Ceci permet de traiter le cas des valeurs factorisées. En effet, s'il y a une ou plusieurs valeurs factorisées parmi les $i-1$ premières composantes, alors il existe plusieurs tuples de taille i construits à partir du même t_{i-1}. Tous ces tuples partageant une partie commune doivent être extraits.

Certains tuples peuvent comporter une valeur manquante pour la composante i. La ligne 6 permet de gérer ce cas. Lorsque aucun des tuples t_i construits à partir du même tuple de taille $i-1$ t_{i-1} n'est extrait (classé positif), la composante i est supposée manquante. Le tuple $(t_{i-1}, \texttt{null})$ est alors ajouté à l'ensemble S^+_i des tuples extraits.

La sortie de l'algorithme 4 est l'ensemble S^+_n des n-uplets extraits du document d'entrée d.

[1] ce point est abordé plus en détail dans la section 4.6.2

Algorithme 4 Algorithme d'extraction

Entrée: un document d ; n classificateurs $c_1, \ldots, c_n$

1: $S_1 = \{(l) \mid l \in L(d)\}$;
2: $S_1^+ = \{t_1 \in S_1 \mid c_1(rep_1(t_1)) = +1\}$
3: **pour** $i = 2$ **à** n **faire**
4: $\quad S_i = \{t_i = (t_{i-1}, l) \mid t_{i-1} \in S_{i-1}^+,\ l \in L(d)\}$
5: $\quad S_i^+ = \{t_i \in S_i \mid c_i(rep_i(t_i)) = +1\}$
6: $\quad S_i^+ = S_i^+ \cup \{(t_{i-1}, \texttt{null}) \mid t_{i-1} \in S_{i-1}^+,\ \forall l \in L(d),\ c_i(rep_i((t_i, l))) = -1\}$
7: **fin pour**

Sortie: S_n^+

4.6.2 Complexité

On peut examiner la complexité de l'algorithme 4 sous deux angles : celui du nombre de tuples extraits, et celui de la complexité de la représentation des exemples.

De part sa nature incrémentale, à chaque étape i du procédé d'extraction le nombre de tuples candidats est borné. Il s'agit du nombre de tuples extraits à l'étape $i-1$ multiplié par le nombre de feuilles de l'arbre [2]. Le pire des cas advient lorsqu'à chaque étape tous les tuples candidats sont extraits. Ainsi le nombre de n-uplets extraits est le nombre de feuilles du document d à la puissance n. Cependant ce cas ne peut survenir que si chacun des classificateurs c_i classe tous les tuples candidats de l'étape i comme positif. On peut raisonnablement espérer qu'une telle séquence de classificateurs ne peut être induite d'après un ensemble de documents d'apprentissage. En pratique, ce cas n'a pas été rencontré au cours de nos différentes expérimentations (voir la section 4.8).

La complexité de l'algorithme 4 dépend aussi de celle du codage à l'aide des fonctions de représentation rep_i. La représentation des tuples peut se faire de manière efficace avec le codage proposé dans la section 4.5. Les attributs portant sur les nœuds et les feuilles sont calculés lors de la lecture du document d (voir 4.5.5). Ils sont donc disponible selon une complexité linéaire en la taille du document d'entrée (en nombre de nœuds). Les attributs portant sur les dépendances entre deux nœuds sont calculés d'une manière paresseuse (*i.e.* uniquement lorsqu'ils le nécessitent). De plus, ils sont calculés seulement pour les tuples extraits lors de l'étape précédente.

Il se peut qu'un classificateur c_i n'utilise qu'un sous-ensemble des attributs du codage des tuples. Si tel est le cas, seuls les attributs concernés sont à calculer au sein de la fonction de représentation rep_i correspondante. Cela réduit le temps de calcul du processus d'extraction, ainsi que sa complexité.

4.6.3 Propriétés

Cette section présente quelques propriétés de notre algorithme, à savoir :
- l'influence des valeurs manquantes ;
- l'influence de l'ordre d'extraction ;

[2] pour $i = 1$ le nombre de singletons candidats est le nombre de feuilles

– l'expressivité en terme de requêtes n-aires.

4.6.3.1 Valeurs manquantes

Les valeurs manquantes ont un double effet sur notre algorithme d'extraction.

D'une part, les tuples qui comportent une valeur manquante pour la composante utilisée comme graine ne peuvent être extraits. En effet, dans l'algorithme 4, l'heuristique de gestion des valeurs manquantes de la ligne 6 est appliquée à partir de la seconde composante. Le principe de cette heuristique est le suivant : si aucun des tuples t_i construits à partir du même tuple t_{i-1}, de taille $i-1$, n'est extrait, la valeur de la composante i est supposée manquante et le tuple $(t_{i-1}, \texttt{null})$ est ajouté à l'ensemble S_i^+ des tuples extraits. Ainsi cette heuristique n'est applicable que pour $i \geq 2$.

D'autre part la composante utilisée comme graine ne doit pas comporter de valeurs manquantes pour aucun n-uplet. Considérons le document de la figure 4.6. Il s'agit d'une liste de morceaux de musique, présentée dans une table, contenant trois enregistrements de la forme *(Titre,Artiste,Album)*. Le premier triplet `(La Cumparsita,Lalo Schifrin,Tango)` est complet. Les deux autres triplets contiennent chacun une valeur manquante (notée `null`), mais pas pour la même composante : `(Tango Apasionado,Astor Piazzola,null)` dont l'album est manquant et `(Sentimientos,null,Tango Nuevo Collection)` dont l'artiste est inconnu. Si la composante *Artiste* est utilisée comme graine, alors le triplet `(Sentimientos,null,Tango Nuevo Collection)` n'est pas extrait. De même si la composante *Album* est utilisée comme graine, alors le triplet `(Tango Apasionado,Astor Piazzola,null)` n'est pas extrait. Par contre l'utilisation de la composante *Titre* comme graine permet l'extraction des trois triplets.

Titre	Artiste	Album
La Cumparsita	Lalo Schifrin	Tango
Tango Apasionado	Astor Piazzola	
Sentimientos		Tango Nuevo Collection

FIG. 4.6 – Document illustrant un des effets des valeurs manquantes sur l'algorithme d'extraction 4

Le pire cas que l'on puisse rencontrer est celui du document de la figure 4.7. Il s'agit d'un annuaire téléphonique qui contient trois enregistrements sous la forme d'un triplet *(nom,prénom,téléphone)*. Cependant, chacun des trois triplets, à savoir `(Atget,Eugène,null)`, `(Gautrand,null,43 72 10 36)` et `(null,Yann,52 62 62)`, contient une valeur manquante pour une composante différente. Ainsi si, par exemple, la composante *nom* est utilisée comme composante graine, alors le triplet `(null,Yann,52 62 62)`, qui n'a pas de valeur pour la composante *nom*, ne sera pas extrait par l'algorithme 4. D'une manière plus générale, quelle que soit la composante utilisée comme graine, l'un des triplets considérés ne sera pas extrait car pour chaque composante il existe un triplet qui n'a pas valeur sur cette composante.

1. – **Nom :** Atget
   – **Prénom :** Eugène
   – **Téléphone** :
2. – **Nom :** Gautrand
   – **Prénom :**
   – **Téléphone** : 43 72 10 36
3. – **Nom :**
   – **Prénom :** Yann
   – **Téléphone** : 52 62 62

FIG. 4.7 – Document illustrant le pire cas des effets des valeurs manquantes sur l'algorithme d'extraction 4

4.6.3.2 Ordre d'extraction pour les organisations de base

L'algorithme d'extraction considère les composantes de la relation à extraire dans un ordre donné. Plusieurs ordres sur les composantes sont envisageables. Parmi eux, certains sont plus optimal que d'autres, en terme de performances d'extraction. Trouver un ordre optimal apparaît ainsi comme un élément critique quant au succès de notre approche. Cependant si l'on se questionne sur l'ordre optimal dans le cas des cinq organisations de base de la section 3.3, on s'aperçoit que l'ordre dans lequel les composantes sont extraites est prépondérant dans le cas d'une organisation des n-uplets avec des composantes factorisées.

En effet, avec une relation n-aire sans composantes factorisées, comme c'est le cas des tables, listes et tables tournées, l'ordre d'extraction peut être, *a priori*, quelconque. Chaque composante permet d'identifier de manière unique le tuple auquel elle appartient. Par exemple, dans une table la position d'une valeur dans une cellule correspond exactement à une ligne et une colonne. Ainsi dans le cas d'une table, la connaissance de la position d'une cellule de la table identifie de manière unique une ligne, c'est-à-dire un tuple. Dans le cas d'une table tournée, les n-uplets sont stockés en colonne et donc la position d'une cellule caractérise un n-uplet.

4.6.3.3 Expressivité en terme de requêtes n-aires

Dans le cas des documents à structure arborescente, une requête n-aire est une fonction de l'ensemble de ces arbres vers l'ensemble des n-uplets de leurs nœuds (pour une définition plus formelle voir la page 137).

Dans le cas unaire, les requêtes exprimables par notre algorithme sont strictement incluses dans les requêtes régulières.

Par contre dans le cas n-aire, notre algorithme permet d'exprimer des requêtes non régulières. Cela est du essentiellement aux attributs qui comparent des branches d'arbres entre elles. Ces attributs permettent l'utilisation de propriétés non exprimables par le biais d'un automate d'arbre et donc par les requêtes régulières.

4.7 Algorithme d'induction

L'algorithme d'induction 5 réalise l'apprentissage d'extracteurs de n-uplets de feuilles fonctionnant selon l'algorithme d'extraction 4. Tout comme de dernier il est incrémental. L'induction d'un extracteur n-aire se fait en n étapes d'apprentissage, à partir d'un corpus de documents complètement annotés en entrée. À chaque étape i de l'algorithme, un classificateur c_i de tuples de longueur i est appris. Ce classificateur est binaire : il étiquette chaque exemple soit par positif, soit par négatif. Les exemples positifs sont la projection des n-uplets à extraire sur les i premières composantes. Devant leur trop grand nombre (exponentiel en n), les exemples négatifs sont sélectionnés selon un procédé inspiré de l'algorithme d'extraction 4. La sortie de l'algorithme 5 est la séquence de n classificateurs $c_1, \ldots, c_n$, qui elle-même est l'entrée de l'algorithme 4. Les propriétés de l'algorithme d'induction sont étudiées, notamment sa complexité et celle de consistance.

4.7.1 Algorithme

L'algorithme 5 est l'algorithme d'induction correspondant à l'algorithme d'extraction 4, et dont le schéma algorithmique est assez similaire.

L'algorithme 5 algorithme est relativement simple. Il prend en entrée un corpus D de documents dans lesquels les n-uplets à extraire sont tous annotés. Tout comme l'algorithme d'extraction 4 le processus d'induction est incrémental. Il comporte une boucle de n étapes d'apprentissage (ligne 1). Chaque apprentissage génère un classificateur c_i de tuples de longueur i.

Dans l'algorithme 5, $S^+(d)$ désigne l'ensemble des n-uplets à extraire du document $d \in D$, et $S_i^+(d)$ la projection de $S^+(d)$ sur les i premières composantes. À chaque étape i, l'ensemble d'apprentissage est obtenu de la manière suivante. L'ensemble des exemples positifs est la projection des n-uplets annotés sur les i premières composantes (ligne 2). S'il existe un tuple t_i dont la valeur de la $i^{\text{ième}}$ composante (notée $t_i[i]$) est `null`, alors t_i n'est pas ajouté à S_i^+. Ainsi le nombre d'exemples positifs est borné par le nombre de tuples dans l'ensemble $\bigcup_{d \in S+(d)}$, c'est à dire le nombre de n-uplets à extraire dans le corpus D.

Par contre le nombre d'exemples négatifs est, à chaque étape, exponentiel en i. En effet les documents sont complètement annotés, donc tout tuple de longueur i qui n'est pas annoté n'est pas à extraire. De fait tous les tuples de feuilles de taille i non annotés sont des exemples négatifs. Pour un document d, le nombre d'exemples négatifs est donc le nombre de feuilles de d exposant i. Il n'est donc pas possible d'utiliser l'intégralité de l'ensemble des exemples négatifs. Seulement un sous-ensemble est sélectionné pour l'apprentissage du classificateur c_i. La sélection des exemples négatifs se fait à la ligne 3, selon un procédé similaire à la génération des tuples candidats dans l'algorithme d'extraction 4. Lors de la première itération ($i = 1$ et $S_0^+ = \emptyset$), les exemples négatifs sont toutes les feuilles du corpus qui ne sont pas annotées comme étant une valeur de la première composante, *i.e.* qui ne sont pas une graine. Le nombre d'exemples négatifs est donc le nombre de feuilles du corpus moins le nombre de graines[3]. Ensuite ($i \neq 1$),

[3]le nombre de graines est au plus le nombre n-uplets à extraire

Algorithme 5 Algorithme d'induction d'un extracteur n-aire

Entrée: un corpus D de documents annotés

1: **pour** $i = 1$ **à** n **faire**
2: $S_i^+ = \bigcup_{d \in D} \{t_i \in S_i^+(d) \mid t_i[i] \neq \texttt{null}\}$
3: $S_i^- = \bigcup_{d \in D} \{t_i = (t'_{i-1}, l) \mid l \in L(d), t_{i-1} \in S_{i-1}^+, t'_i \notin S_i^+\}$
4: $c_i = A(\bigcup_{x \in S_i^+} \{rep_i(x)\}, \bigcup_{x \in S_i^-} \{rep_i(x)\})$
5: **fin pour**

Sortie: la séquence $(c_1, \ldots, c_n)$ de classificateurs

un exemple négatif est obtenu par l'ajout d'une feuille texte l comme $i^{\text{ième}}$ composante à t_{i-1}, un tuple positif de taille $i-1$. On obtient ainsi le tuple de taille i $t'_i = (t_{i-1}, l)$. t'_i est considéré comme un exemple négatif s'il n'existe pas de tuple $t''_i \in S_i^+$ tel que t'_i et t''_i soient identiques. Autrement dit, le tuple t'_i n'est pas la projection d'un tuple à extraire sur les i premières composantes. Ainsi le nombre d'exemples négatifs est borné par le nombre d'exemples positifs dans S_{i-1}^+ multiplié par le nombre de feuilles du corpus.

L'apprentissage du classificateur c_i se fait à la ligne 4 à l'aide de l'apprenant A. Nous avons choisi d'utiliser le même algorithme d'apprentissage quel que soit i. Les représentations de exemples sont calculées à l'aide des mêmes fonctions de codage rep_i que dans l'algorithme d'extraction 4. La sortie de l'algorithme d'induction est la séquence de classificateurs $c_1, \ldots, c_n$, qui est l'entrée de l'algorithme 4.

4.7.2 Consistance

La consistance est une propriété importante. Nous avons déjà vu la consistance d'un classificateur à la section 2.2.1 page 35. Nous définissons maintenant la consistance d'un extracteur.

Un extracteur est consistant avec son corpus d'apprentissage s'il extrait correctement les données de chaque document de ce corpus.

Définition 4.1 (Consistance d'un extracteur) *Soit w l'extracteur induit à partir du corpus D. w est consistant si et seulement si pour tout $d \in D$ on a $w(d) = e(d)$.*

Un extracteur consistant possède une précision, un rappel et une couverture maximum lorsqu'il est appliqué à son corpus d'apprentissage.

Nous montrons ici que l'extracteur obtenu par l'algorithme d'induction 5 est consistant si tous les classificateurs qui le constituent sont également consistants (et inversement).

Propriété 4.1 *L'extracteur w dont le procédé d'extraction est décrit par l'algorithme 4 et dont les classificateurs $c_1, \ldots, c_n$ sont issus de l'algorithme 5 est consistant si et seulement si tous les classificateurs c_i sont consistants.*

Preuve :
Dans le souci d'alléger les notations, les fonctions de représentation rep_i seront omises.

Dans un premier temps, on se place dans le cas où tous les classificateurs c_i sont consistants. Soit D le corpus d'apprentissage et d un document de D. Pour tout i, c_i est consistant avec $S_i^+ \cup S_i^-$. En particulier, on a c_i consistant avec $S_i^+(d) \cup S_i^-(d)$, où $S_i^-(d)$ est l'ensemble des tuples négatifs du document d de longueur i obtenus par le procédé de calcul de la ligne 3 de l'algorithme 5. Deux cas sont alors possibles : soit aucun tuple de $S_n^+(d)$ n'admet de valeur manquante, soit il existe au moins un tuple de $S_n^+(d)$ avec au moins une valeur manquante.

Supposons qu'il n'y a pas de valeurs manquantes parmi les n-uplets $S_n^+(d)$ à extraire de d. Si l'on applique l'extracteur w à d, les tuples de longueur i extraits à chaque étape i de l'algorithme d'extraction 4 sont exactement les tuples $S_i^+(d)$. Notamment, on a $w(d) = S_n^+(d)$, et donc w est consistant.

Supposons maintenant qu'il existe un tuple $t \in S_n^+(d)$ qui comporte une valeur manquante pour la composante j , $1 < j \leq n$ [4]. Jusqu'à l'étape $j-1$ il n'y a pas de valeur manquante. Donc le tuple t_{j-1}, projection de t sur les $j-1$ premières composantes, est extrait en raison de la consistance des $c_1, \ldots, c_{j-1}$. Le classificateur c_j classe tous tuples de taille j construits à partir de t_{j-1} comme négatif. En effet, lors de l'apprentissage de c_j, comme la valeur de la composante j est `null`*, t_j n'est pas un exemple positif, et pour tout $l \in L(d)$ le tuple (t_{j-1}, l) est un exemple négatif. À l'étape j, les tuples de taille j candidats à l'extraction et construits sur t_{j-1}, sont exactement les exemples négatifs de la forme $(t_{j-1}, l), l \in L(d)$. Comme c_j est consistant, tous ces tuples sont classés négatifs à la ligne 5 de l'algorithme 4. Ainsi la condition de la ligne 6 du même algorithme est vérifiée, et le tuple $t_j = (t_{j-1},$* `null`*$)$ est extrait. Les composantes $j+1$ à n ne présentent pas de valeurs manquantes, et comme les classificateurs $c_{j+1}, \ldots, c_n$ sont consistants, les tuples t_{j+1} à $t_n = t$ sont successivement extraits. Donc le tuple t est extrait malgré une valeur manquante, et w est consistant. Ce résultat s'étend facilement au cas où t possède au plus $n-1$ valeurs manquantes (sauf la première composante). Donc l'extracteur w est consistant.*

Dans un deuxième temps, nous étudions les cas où l'extracteur w est consistant. Soit un document d du corpus D. Supposons qu'il existe un classificateur $c_j, 1 \leq j \leq n$ non consistant. Lorsqu'on applique w à d, il existe alors un tuple t_j candidat à l'extraction tel que $c_j(t_j)$ soit différent de l'annotation de t_j dans le document d. Si t_j est à extraire, alors $c_j(t_j)$ est négatif et t_j n'est pas extrait. Si t_j n'est pas extrait alors qu'il aurait du l'être, alors il existe au moins un n-uplet construit sur t_j qui n'est pas extrait. Et donc w n'est pas consistant. Ce qui est absurde, donc tous les classificateurs c_i sont consistants. ◄

4.7.3 Implémentation

Les algorithmes d'extraction 4 et d'induction 5 font l'objet d'une implémentation au sein d'un système d'extraction d'information n-aire pour les documents `HTML` et `XML`. Il s'agit du système PAF qui permet à la fois l'apprentissage d'un extracteur n-

[4] l'algorithme d'extraction impose qu'aucun tuple ne possède de valeur nulle pour la première composante ; de plus le n-uplet (`null`, ..., `null`) n'est pas considéré comme étant à extraire

aire à partir d'un ensemble de documents complètement annotés et l'application de cet extracteur à un ensemble de documents pour en extraire des données.

La prochaine section présente l'évaluation expérimentale que nous avons menée sur PaF.

4.8 Expériences

Cette section présente l'évaluation expérimentale de notre approche.

Sur des jeux de données classiques en extraction d'information, en ceux du corpus Rise, nous obtenons des performances supérieures ou égales aux systèmes classiques décrits dans la section A. Mais ces corpus ne correspondent plus à l'état actuel du Web. En effet, ils datent de 1997 et sont donc représentatifs des standards du Web de l'époque. Ils sont trop simples et les seules organisations des données qu'ils comportent sont des listes et des tables simples.

Pour évaluer notre approche les organisations des données identifiées dans le chapitre 3, nous utilisons le corpus artificiel DataFoot. Les documents de ce corpus sont produits par programme, d'après une base de données, selon les cinq cas de base mis en évidence dans le chapitre 3.

Nous avons également mené l'évaluation de notre système sur des documents `XHTML` représentatifs du Web actuel. Pour cela nous avons constitué le corpus CorporateData avec de pages issues de sites Web de statistiques, de météorologie,... qui proposent aussi des données n-aire disposées selon toutes les organisations identifiées dans le chapitre 3. Notre approche permet de traiter tous ces cas, alors que les approches classiques échouent.

Avant de présenter les différents résultats expérimentaux, nous décrivons dans la section suivante les mesures d'évaluation employées.

4.8.1 Évaluation

Cette section expose la méthodologie d'évaluation d'un algorithme d'induction supervisée d'extracteurs. Pour cela on évalue la qualité des extractions produites par l'extracteur induit par l'algorithme évalué. Cette évaluation suppose l'utilisation d'un corpus de documents complètement annotés : toutes les données à extraire sont connues. Nous donnons la définition des trois mesures classiques d'évaluation des performances, à savoir la précision, le rappel et la F-mesure, ainsi que la mesure de couverture.

La précision, le rappel et la F-mesure, qui est la moyenne harmonique de la précision est du rappel, sont des mesures provenant du domaine de la recherche d'information. Elles sont également d'un usage courant en extraction d'information. Avant de définir ses mesures, il est nécessaire de préciser comment évaluer la correction d'une donnée extraite. Plusieurs critères sont envisageables pour cela. Nous illustrons cette diversité en examinant d'abord le cas de l'extraction unaire puis le cas n-aire.

4.8.1.1 Correction des données extraites dans le cas unaire

[10] précise comment évaluer la correction des données dans le cas unaire pour des documents représentés sous la forme d'une séquence. Dans ce cas l'extraction unaire revient à extraire un ensemble de séquences, chacune d'entre elle étant une valeur extraite. Bien que cet article considère uniquement le cas des documents non-structurés, les trois critères proposés pour l'évaluation de la correction d'une séquence extraite sont également applicables aux documents semi-structurés représentés sous la forme d'une séquence.

Le premier d'entre eux est le plus strict : une donnée extraite est correcte si elle correspond exactement, au caractère près, à l'une des séquences à extraire. Le second critère est une relaxation du premier : une donnée extraite est correcte si elle contient une donnée à extraire (dans son intégralité). Enfin, le dernier critère est encore plus relâché : une donnée extraite est correcte si elle contient une partie d'une donnée à extraire.

Dans le cas des documents semi-structurés représentés comme des arbres, l'extraction unaire est l'extraction d'un ensemble de nœuds. L'extraction d'un nœud est correcte si ce nœud appartient à l'ensemble des nœuds à extraire. Dans ce cas l'extraction unaire peut-être vue comme la décoration de chaque nœud de l'arbre du document par l'étiquette "à extraire" ou "à ne pas extraire". Une extraction est correcte si l'étiquette décorant le nœud est celle attendue.

Dans cette thèse, nous considérons uniquement le critère de correction le plus strict : une donnée extraite est correcte si elle égale à l'une des valeurs à extraire.

4.8.1.2 Correction des données extraites dans le cas n-aire

L'extraction n-aire consiste à extraire un ensemble de n-uplets des documents d'entrée, quelque soit leur nature. Définir la correction d'un n-uplet extrait nécessite de choisir un critère de correction pour chaque composante du n-uplet. En pratique, le même critère est utilisé pour évaluer si la valeur d'une composante est correcte ou pas.

[10] ne propose pas de critères pour définir la correction d'un n-uplet. Dans le cadre de cette thèse, on considère que l'extraction d'un n-uplet est correct si toutes ses composantes le sont en qu'elles sont bien toutes correctement associées.

4.8.1.3 Précision, Rappel et F-mesure

Les définitions des mesures de précision, rappel et F-mesure sont identiques pour le cas unaire et le cas n-aire. Soit w un extracteur et D un corpus de documents. Pour $d \in D$, on note $w(d)$ les informations extraites du document d par w et $e(d)$ les informations qui sont réellement à extraire de d. On définit aussi l'ensemble des informations extraites du corpus D par :

$$W = \bigcup_{d \in D} w(d)$$

et l'ensemble des informations à extraire de D par :

$$E = \bigcup_{d \in D} e(d)$$

$W \cap E$ est alors l'ensemble des données extraites correctement.

Définition 4.2 (Précision) *La précision de l'extracteur w est définie par :*

$$P = \frac{|W \cap E|}{|W|}$$

Définition 4.3 (Rappel) *Le rappel de l'extracteur w est défini par :*

$$R = \frac{|W \cap E|}{|E|}$$

Par définition, ces deux mesures prennent leur valeur entre 0 et 1, mais par commodité on les exprime généralement sur une échelle de 0 à 100. La précision évalue le ratio d'informations à extraire parmi les informations extraites. La précision vaut 100 si toutes les données extraites devaient bien l'être. Le rappel mesure la quantité d'informations extraites correctement parmi l'ensemble des informations à extraire. Un rappel de 100 signifie que l'extracteur a bien reconnu toutes les valeurs à extraire. La précision mesure la qualité des extractions tandis que le rappel met en évidence la proportion d'extractions correctes. Une précision et un rappel de 100 indique un extracteur parfait, dans le sens où il ne commet aucune erreur sur le corpus considéré.

Définition 4.4 (F-mesure) *Soit P la précision et R le rappel de l'extracteur w. La F-mesure de w est définie par :*

$$F = \frac{2PR}{P + R}$$

La F-mesure évalue la qualité globale d'un extracteur en combinant sa précision et son rappel en une mesure unique.

4.8.1.4 Couverture

La couverture d'un extracteur est le pourcentage de documents dont toutes les informations sont correctement extraites.

Définition 4.5 (Couverture) *La couverture du corpus D par l'extracteur w est définie par :*

$$C = \frac{|\{d \in D \mid w(d) = e(d)\}|}{|D|}$$

Une couverture de 100 signifie que l'extracteur traite correctement chaque document.

4.8.1.5 Protocole expérimental

Le protocole expérimental utilisé dans toutes les expériences de ce chapitre est identique et se déroule selon le scénario suivant. 10 documents sont tirés aléatoirement de chaque corpus. Chaque expérience est menée par une technique de validation croisée : les 10 documents sont répartis en k blocs. L'apprentissage est réalisé sur un bloc tandis que le test, *i.e.* l'évaluation de l'extraction, se fait sur les autres blocs. Ce processus est répété de telle sorte que chacun des blocs serve en apprentissage. La technique de validation croisée que nous utilisons est inversée par rapport à la technique habituelle qui utilise $k-1$ blocs en apprentissage et le bloc restant en test.

En pratique et sauf mention contraire, la valeur de k est soit 5 soit 10. Ainsi la taille des blocs est soit 2 soit 1. Autrement dit, l'apprentissage est réalisé avec 2 documents (respectivement 1 document) et le test avec 8 documents (respectivement 9 documents). Le choix d'utiliser au plus 2 documents pour l'apprentissage est motivé par la conviction que l'induction d'un extracteur peut être réalisée à partir du plus petit nombre possible d'annotations.

Les performances de l'extracteur induit à partir des documents d'apprentissage sont évaluées à l'aide des mesures de précision (notée P), de rappel (notée R), de F-mesure (notée F) et de couverture[5] (notée C) sur les documents de test.

4.8.2 Expériences sur le corpus DataFoot

Cette section présente les expériences réalisées sur le corpus DataFoot, qui est un corpus artificiel créé d'après une base de données (de résultats des championnats de football français). La particularité de ce corpus est de présenter les mêmes données selon les cinq organisations des tuples identifiées dans le chapitre 3.

Le but des expériences sur le corpus DataFoot est d'évaluer notre approche de l'extraction n-aire sur les organisations de base. Les questions testées expérimentalement sont les suivantes :

- quelle est l'impact de l'apprenant supervisé de base de l'algorithme 5 sur les performances de l'extracteur induit ?
- quelle est l'efficacité de l'heuristique employée dans l'algorithme 4 pour gérer les valeurs manquantes ?

4.8.2.1 Description du corpus

Le corpus DataFoot est un corpus artificiel construit par programme depuis une base de données des résultats du championnat de football français. Le corpus DataFoot est constitué de cinq groupes de documents. Chaque groupe correspond à une des organisations de bases, à savoir :

- liste ;
- table ;
- table tournée ;
- factorisation ;

[5] ces mesures sont présentées dans le chapitre 1 à la section 4.8.1 page 97

Organisation	Arité	Nb. de tuples	Nb. de feuilles textes	Nb. de nœuds	Hauteur	Branchement
Table	4	9.20 (2.89)	182.60 (37.59)	384.40 (78.07)	7.00 (0.00)	2.11 (3.85)
Table tournée	4	9.20 (2.89)	182.60 (37.59)	387.20 (75.18)	7.00 (0.00)	2.09 (3.67)
Liste	4	9.20 (2.89)	261.60 (66.50)	558.80 (141.68)	6.00 (0.00)	2.01 (7.75)
Factorisation	4	9.20 (2.89)	223.40 (39.90)	477.50 (83.87)	9.00 (0.00)	2.04 (3.47)
Table croisée	3	16.50 (5.68)	32.50 (7.09)	99.00 (14.88)	7.00 (0.00)	2.02 (2.13)

TAB. 4.1 – Description statistique du corpus DATAFOOT

– table croisée.

Pour les quatre premiers groupes, la relation n-aire cible est la relation quaternaire (*club*,*season*,*round*,*date*). Il existe deux versions des quatre premiers groupes : l'une sans valeurs manquantes, et l'une où 37,5% des valeurs de la composante *date* sont manquantes. La relation cible du dernier groupe, cas des tables croisées, est la relation ternaire (*win*,*club1*,*club2*).

Chaque groupe contient 20 documents `XHTML` qui eux-mêmes contiennent entre 5 à 15 enregistrements issus de la base de données. Dans le cas de quatre premiers groupes (liste, table, table tournée et factorisation), les mêmes ensembles d'enregistrements sont utilisés pour produire les mêmes fichiers, la seule différence étant la manière de stocker ces enregistrements dans l'arbre `XHTML` des documents.

Le tableau 4.1 donne une description statistique du corpus DATAFOOT. De gauche à droite, les colonnes de ce tableau fournissent les informations suivantes :

– l'organisation ;
– l'arité de la relation n-aire cible ;
– le nombre moyen de tuples par document ;
– le nombre moyen de feuilles texte par document ;
– le nombre moyen de nœuds par document ;
– la hauteur moyenne de l'arbre représentant un document ;
– le facteur de branchement moyen de l'arbre représentant un document.

Les chiffres indiqués entre parenthèses sont les écarts-types. On peut constater que les arbres des documents `XHTML` du corpus DATAFOOT sont peu profonds et peu larges. Le nombre de tuples moyen par document est de 9 pour les quatre premières organisations et de 16 pour les tables croisées.

4.8.2.2 Impact de l'algorithme de classification supervisée

Le but de cette expérience est d'évaluer l'impact de l'algorithme de classification supervisée, utilisé comme brique d'apprentissage de base, sur les performances de l'extracteur induit par l'algorithme 5. Pour ce faire, plusieurs algorithmes de classification supervisée sont comparés dans le rôle de l'apprenant A de l'algorithme 5.

Cette expérience est réalisée avec les trois apprentis supervisés suivants :

– DLG [85] qui est un algorithme d'apprentissage par couverture à base de moindres généralisés ;
– ADABOOSTMG [83] est également fondé sur les moindres généralisés mais l'algorithme d'apprentissage est ADABOOST [34, 77] ;

Organisation	P	R	F	C
Table	100	100	100	100
Table tournée	87	73.19	78.62	30
Liste	100	71.03	82.39	22.50
Factorisation	95.94	90.76	92.96	62.50
Table croisée	51.15	44.94	47.55	35

TAB. 4.2 – Résultats sur le corpus DATAFOOT pour DLG

Organisation	P	R	F	C
Table	100	100	100	100
Table tournée	84.85	84.85	84.85	70
Liste	100	76.00	85.33	40
Factorisation	100	100	100	100
Table croisée	58.83	58.83	58.83	67.50

TAB. 4.3 – Résultats sur le corpus DATAFOOT pour ADABOOSTMG

– C5.0 [73] est une amélioration de l'algorithme d'induction d'arbres de décision C4.5 [71].

Le protocole expérimental est celui décrit à la section 4.8.1.5. Le nombre de blocs de validation croisée k est fixé à 5, c'est à dire que l'apprentissage se fait avec 2 documents et le test avec 8 (le nombre de documents au total étant de 10).

Les tableaux 4.3, 4.2 et 4.4 donnent respectivement les résultats obtenus avec DLG, ADABOOSTMG [6] et C5.0. On constate que les meilleurs résultats sont obtenus avec C5.0, dont la F-mesure varie entre 92.6 et 100. Les faibles résultats de DLG s'expliquent en raison de sa stratégie gloutonne pour la construction d'un classificateur. Il est important de noter que ADABOOSTMG peut égaler C5.0 en augmentant le nombre d'étapes de boosting, cependant le temps de calcul devient prohibitif. L'algorithme C5.0 est à la fois performant et rapide (l'apprentissage et la classification ne prennent que quelques secondes).

Dans la suite de cette thèse, C5.0 sera systématiquement l'algorithme d'apprentissage supervisé utilisé comme apprenant de base dans nos algorithmes.

4.8.2.3 Valeurs manquantes

Le but de cette expérience est de montrer l'efficacité de l'heuristique, notée h_vm, employée dans l'algorithme d'extraction 4 pour gérer les valeurs manquantes. C'est la version du corpus DATAFOOT avec valeurs manquantes que est utilisée ici (qui contient les quatre premières organisations : liste, table, table tournée et factorisation). Elle comporte 37,5% de valeurs manquantes pour la composante *date*.

[6] les résultats de ADABOOSTMG sont obtenus après 50 étapes de boosting

Organisation	P	R	F	C
Table	100	100	100	100
Table tournée	100	100	100	100
Liste	86.76	100	92.90	10
Factorisation	100	100	100	100
Table croisée	100	100	100	100

TAB. 4.4 – Résultats sur le corpus DATAFOOT pour C5.0

Organisation	P	R	F	C
Table	100	100	100	100
Table tournée	81.90	94.85	83.36	85
Liste	89.32	100	94.35	40
Factorisation	100	100	100	100

TAB. 4.5 – Résultats sur le corpus DATAFOOT avec l'heuristique des valeurs manquantes

L'expérience consiste à évaluer l'extracteur induit avec et sans l'heuristique de gestion des valeurs manquantes. Le tableau 4.5 présente les résultats de l'extracteur induit par l'algorithme 4, tandis que le tableau 4.6 donne les résultats obtenus avec l'algorithme 4 privé de la ligne 6 [7].

Si l'on compare les résultats obtenus avec et sans la gestion des valeurs manquantes, on s'aperçoit son efficacité. En effet, lorsque la gestion des valeurs manquantes est activée, la F-mesure varie entre 83.36 et 100, tandis que lorsqu'elle est inactive, la F-mesure chute entre 61.84 et 73.98. On constate ainsi du gain apporté par l'heuristique $h_v m$ Cette différence de performances est d'autant plus flagrante si l'on compare la mesure de couverture : alors qu'elle varie entre 40 et 100 dans le tableau 4.5 elle chute à 0 pour toutes les organisations dans le tableau 4.6, ce qui signifie que dans aucun document tous les tuples sont correctement extraits.

On peut remarquer que la précision n'est pas affectée par l'activation ou pas de l'heuristique $h_v m$. Cependant, le rappel est fortement touché : il passe, en ordre de grandeur, de 100 avec $h_v m$ à environ 59 sans $h_v m$. Cette chute illustre bien que $h_v m$ permet de récupérer les tuples avec valeurs manquantes qui ne sont pas extraits par le procédé de classification.

4.8.3 Expériences sur le corpus Rise

RISE [8] est un ensemble de corpus d'évaluation standard en extraction d'information. L'inconvénient majeur de RISE est d'être distribué sans annotation. L'évaluation de PAF sur RISE a nécessité une annotation manuelle des documents. C'est pourquoi

[7] voir page 90

[8] disponible à http://www.isi.edu/info-agents/RISE/index.html

Organisation	P	R	F	C
Table	100	58.69	73.96	0
Table tournée	81.54	58.71	61.84	0
Liste	100	58.71	73.98	0
Factorisation	100	58.71	73.98	0

TAB. 4.6 – Résultats sur le corpus DATAFOOT sans l'heuristique des valeurs manquantes

Corpus	Arité	Nb. de tuples	Nb. de feuilles textes	Nb. de nœuds	Hauteur	Branchement
BIGBOOK	2	18.29 (4.86)	109.52 (28.44)	503.98 (116.50)	8.00 (0.00)	1.57 (2.16)
OKRA	3	13.23 (15.98)	119.88 (129.77)	360.91 (387.82)	8.00 (0.00)	1.63 (3.25)
S20	4	32.70 (11.70)	279.60 (93.57)	588.20 (187.15)	10.00 (0.00)	2.29 (4.59)
S1	3	40.30 (2.10)	341.90 (35.57)	756.60 (104.12)	9.50 (4.50)	3.38 (24.41)
IAF	6	9.00 (2.41)	113.40 (24.80)	449.70 (103.45)	10.00 (0.00)	1.86 (3.89)

TAB. 4.7 – Description statistique du corpus RISE

seuls cinq des corpus de RISE ont été utilisés.

Ajoutons à cela que les données des corpus de RISE sont organisées soit dans une table soit sous forme de liste, et que les sites Web dont sont issus ces corpus datent de 1997. Ils correspondent donc aux standards du Web de l'époque.

Néanmoins, l'intérêt de RISE est de mener des expérimentations afin de comparer PAF aux autres systèmes d'extraction d'information.

4.8.3.1 Description du corpus

Le tableau 4.7 donne une description statistique du corpus RISE, comme celle présentée sur DATAFOOT à la section 4.8.2.1. On peut constater que les arbres des documents `HTML` de RISE sont légèrement plus profonds que ceux du corpus DATAFOOT. Les documents de RISE sont aussi plus gros car ils contiennent plus de nœuds.

Parmi les corpus disponibles dans RISE, les cinq retenus sont les suivants :

- BIGBOOK avec la relation cible (*name*,*address*) ;
- OKRA avec la relation cible (*name*,*email*,*score*) ;
- S20 avec la relation cible (*file*,*size*,*type*,*score*) ;
- S1 avec la relation cible (*company*, *price*, *product*) ;
- IAF. avec la relation cible (*name*, *email*, *lastupdate*, *organization*, *altname*, *serviceprovider*).

Sur le corpus IAF, trois des six composantes de la relation cible comportent des valeurs manquantes.

4.8.3.2 Résultats

Le tableau 4.8 présente les résultats expérimentaux. De gauche à droite, les colonnes de ce tableau fournissent les informations suivantes :

Corpus	Organisation	D	P	R	F	C
BIGBOOK	Table	1	100	100	100	100
OKRA	Table	1	100	100	100	100
S20	Liste	1	100	100	100	100
S1	Liste	1	88.65	88.57	88.61	80.1
IAF	Liste	2	61.51	67.66	64.22	61.2

TAB. 4.8 – Résultats expérimentaux sur le corpus RISE

- le nom du corpus ;
- l'organisation ;
- le nombre de documents utilisés pour l'apprentissage ;
- puis les mesures de précision, de rappel, de F-mesure et de couverture, notées respectivement P, R, F, et C.

Sur BIGBOOK, OKRA et S20, PAF obtient 100 en F-mesure. Pour le problème S1 PAF n'obtient pas 100 en F-mesure car pour une extraction parfaite il est nécessaire de tester la présence de certains mots à l'intérieur des feuilles textes.

Le tableau 4.9 donne une comparaison de PAF, WIEN et LIPX sur RISE. La comparaison avec STALKER n'est pas possible, car dans [65] ce dernier n'est pas évalué sur l'extraction n-aire. Il en est de même pour SOFT MEALY [42] dont les performances ne sont évaluées qu'à l'aide du rappel. Enfin, précisons que les résultats sont extraits de [81], qui lui même les a extraits des articles correspondants. Cette démarche s'explique par le fait que la plupart de ces systèmes ne sont pas disponibles, ce qui ne permet pas une confrontation expérimentale réaliste, réalisée dans les mêmes conditions. Ainsi cette comparaison donnée à titre indicatif.

À la lecture du tableau 4.9, on constate que :

- PAF est supérieur à LIPX sur les corpus BIGBOOK, OKRA et IAF [9] ;
- PAF atteint les performances de WIEN sur les corpus BIGBOOK, OKRA et S20 ;
- PAF est le meilleur des trois systèmes sur IAF, qui le plus dur des cinq corpus : PAF est supérieur à LIPX (qui obtient 32 de F-mesure) et à WIEN dont l'algorithme d'induction n'est pas applicable à cause des valeurs manquantes.

Dans le cas n-aire, il n'est pas possible de comparer PAF à STALKER car ce dernier utilise n extracteurs unaires pour traiter l'extraction n-aire et il n'est évalué que dans le cas unaire et pas sur la re-combinaison des tuples. Cependant, sur les tâches d'extraction d'information n-aires que l'on peut définir sur RISE, PAF obtient les performances de STALKER (sauf sur S1 comme mentionné précédemment). En particulier sur IAF, la F-mesure de PAF varie de 84 à 100, alors qu'il obtient 64 dans le cas n-aire. Ces résultats illustrent que sur le corpus IAF la reconstruction des tuples est plus difficile que l'extraction des composantes.

[9] les résultats de LIPX ne sont pas disponibles pour les corpus S1 et S20

Corpus	WIEN			LIPX			PAF		
	P	*R*	*F*	*P*	*R*	*F*	*P*	*R*	*F*
BIGBOOK	100	100	100	100	89.3	94.4	100	100	100
OKRA	100	100	100	100	88.1	93.7	100	100	100
S20	100	100	100	-	-	-	100	100	100
S1	100	100	100	-	-	-	88.65	88.57	88.61
IAF	non applicable			84	19.4	32	61.51	67.66	64.22

TAB. 4.9 – Comparaison des systèmes sur le corpus RISE

Corpus	Arité	Nb. de tuples	Nb. de feuilles textes	Nb. de nœuds	Hauteur	Branchement
BBC	4	5.00 (0.00)	195.00 (0.00)	763.00 (0.00)	14.00 (0.00)	1.91 (2.84)
EXCITE 1	5	5.00 (0.00)	259.40 (0.49)	1222.70 (2.57)	29.00 (0.00)	1.49 (2.53)
EXCITE 2	5	6.00 (0.00)	259.40 (0.49)	1222.70 (2.57)	29.00 (0.00)	1.49 (2.53)
BLS	3	29.00 (28.58)	235.40 (34.31)	579.70 (74.33)	15.00 (0.00)	1.95 (5.10)
BLS 2	3	3.56 (1.17)	95.11 (2.33)	295.00 (4.59)	15.00 (0.00)	1.89 (2.29)
BLS 3	3	24.00 (0.00)	132.00 (0.00)	369.00 (0.00)	15.00 (0.00)	1.99 (2.80)
BLS 4	3	6.00 (0.00)	509.60 (14.38)	1345.20 (41.19)	15.00 (0.00)	1.79 (3.65)
BEA 1	4	8.30 (5.10)	128.00 (30.64)	445.10 (77.05)	16.00 (0.00)	1.66 (2.23)
BEA 2	3	2.60 (0.66)	78.00 (2.14)	313.20 (4.49)	16.00 (0.00)	1.62 (2.12)
BTS	3	20.00 (0.00)	382.00 (0.00)	1041.00 (0.00)	20.00 (0.00)	1.74 (3.35)

TAB. 4.10 – Description statistique du corpus CORPORATEDATA

4.8.4 Expériences sur le corpus CorporateData

CORPORATEDATA est constitué d'un ensemble de corpus issus de sites Web contemporains, comme des sites de statistiques ou de météorologie. Les organisations les plus difficiles, comme les tables tournées, croisées et les organisations avec factorisation, sont majoritaires dans CORPORATEDATA.

4.8.4.1 Description du corpus

Le tableau 4.10 présente la description statistique du corpus CORPORATEDATA. On peut constater que les arbres des documents `XHTML` de CORPORATEDATA sont beaucoup plus volumineux, à la fois en terme de contenu et de structure, que ceux des corpus DATAFOOT et RISE. Ils sont également plus profonds.

4.8.4.2 Résultats

Le tableau 4.11 présente les résultats de PAF sur le corpus CORPORATEDATA. La troisième colonne indique le nombre de documents d'apprentissage, noté D.

On peut constater que PAF permet d'obtenir des extracteurs parfaits sur la plupart des corpus à partir d'un à deux documents complètement annotés en apprentissage. Deux pages sont nécessaires lorsqu'il y a une importante variation dans la taille des tables ou dans le nombre de tuples à extraire d'une page à l'autre.

Corpus	Organisation	D	*P*	*R*	*F*	*C*
BBC	Factorisation, table	1	100	100	100	100
Excite 1	Factorisation, table tournée	1	100	100	100	100
Excite 2	Factorisation, table, table tournée	1	100	100	100	100
bls	Table tournée	1	100	100	100	100
bls 2	Table croisée	1	100	100	100	100
bls 3	Table croisée	2	100	90	93.33	80
bls 4	Table tournée	1	100	100	100	100
bea 1	Factorisation, table tournée	2	97.11	88.58	92.58	82.5
bea 2	Table croisée	2	86.47	100	89.78	72.5
bts	Table croisée	1	100	100	100	100

Tab. 4.11 – Résultats sur le corpus CorporateData

4.9 Conclusion

Ce chapitre expose notre approche de l'extraction n-aire ainsi que l'algorithme d'induction associé. L'apport essentiel de notre approche est la proposition d'un algorithme d'apprentissage interactif d'extracteurs n-aires indépendant de l'organisation des données. PAF, notre système d'extraction n-aire, travaille sans aucun post-traitement car il réalise l'extraction des composantes et la construction des tuples simultanément. Il permet en outre d'extraire à partir d'organisations diverses comme les tables tournées ou croisées et traite le cas des valeurs manquantes.

L'algorithme d'extraction sous-jacent possède deux caractéristiques clefs :

- il est *incrémental* : les tuples sont extraits par taille croissante : d'abord les singletons, ensuite les couples, *etc.* et ce jusqu'à l'obtention des n-uplets ;
- il utilise une procédure d'enrichissement de la représentation : les tuples de longueur i sont codés avec la connaissance des tuples extraits de longueur $i-1$.

Dans notre système, un extracteur est une séquence de n classificateurs binaires c_i. Chaque classificateur c_i assure l'extraction des tuples de taille i. Les tuples sont représentés par un codage attribut-valeur décrivant d'une part le contexte local des composantes d'un tuple et d'autre part les relations entre les différentes composantes.

Nous proposons également l'algorithme d'apprentissage associé à notre procédé d'extraction. Notre algorithme d'apprentissage est itératif et consiste à apprendre la séquence des n classificateurs c_i à l'aide de techniques standards d'apprentissage (en particulier chaque c_i est un arbre de décision obtenu à l'aide de C5.0 [73]) à partir de documents complètement annotés.

PAF n'est pas spécifique à un type d'organisation des données mais permet de traiter des cas complexes comme les tables tournées ou croisées, sur lesquelles les approches existantes échouent. Ce chapitre démontre cela à travers des expérimentations : PAF atteint les performances des systèmes existants sur les organisations simples et réussit également sur des organisations plus complexes sur lesquels les précédents systèmes obtiennent de piètres résultats. De plus, les expériences menées montrent que l'algorithme d'induction de PAF permet d'obtenir des extracteurs performants à partir de peu de

documents complètement annotés. Enfin, précisons que les processus d'extraction et d'apprentissage sont rapides.

Chapitre 5

Induction interactive d'extracteurs

5.1 Introduction

L'algorithme 5 (exposé à la section 4.7 du chapitre 4) accepte en entrée des documents complètement annotés, *i.e.* dans lesquels toutes les informations à extraire sont marquées. Cette contrainte est trop forte pour un utilisateur final, surtout dans le cas de documents contenant beaucoup de tuples. Afin de s'en soustraire et de réduire le nombre d'annotations de l'utilisateur, nous proposons dans ce chapitre d'induire un extracteur sans nécessairement annoter complètement les documents d'apprentissage. Pour ce faire, nous plaçons l'utilisateur dans un cadre interactif en le faisant dialoguer avec le programme d'apprentissage. Le principe du cadre interactif est que l'utilisateur annote partiellement les documents d'apprentissage jusqu'à l'induction d'un extracteur capable d'annoter complètement et correctement lesdits documents. Notre objectif est ici d'obtenir un système interactif d'induction d'extracteurs n-aires fonctionnant à partir d'un faible nombre d'interactions. Les interactions envisagées doivent également être le plus simple possible pour que le système soit utilisable aisément par tout utilisateur.

L'approche interactive de l'induction que nous proposons dans ce chapitre est inspirée de celle de Squirrel [12], que nous étendons ici au cas n-aire. Cette approche suppose que les interactions ont lieu document par document. Elle se déroule selon le scénario suivant. L'utilisateur choisit un document dc, appelé document courant, et annote quelques tuples à extraire. Cette annotation partielle sert d'amorce à l'apprentissage d'un extracteur. Ce dernier est ensuite appliqué à dc afin d'en proposer une annotation complète qui est soumise à l'évaluation de l'utilisateur. L'annotation proposée par l'extracteur préserve l'annotation partielle de l'utilisateur. Si l'utilisateur juge l'annotation de l'extracteur correcte, le processus d'induction se termine ou se poursuit avec un autre document. Sinon, en fonction des erreurs commises par l'extracteur courant, l'utilisateur annote d'autres exemples, non extraits, et/ou des contres-exemples, *i.e.* des tuples extraits alors qu'ils n'auraient pas dus l'être. Cette nouvelle annotation enrichit l'annotation initiale et sert à relancer l'induction d'un nouvel extracteur. Ce processus interactif est itéré jusqu'à l'obtention d'un extracteur jugé satisfaisant par

l'utilisateur.

Ainsi notre scénario interactif comporte deux boucles :
- la boucle de parcours des documents d'apprentissage ;
- la boucle d'interactions sur le document courant.

Afin de suivre l'approche itérative d'induction développée dans le chapitre précédent, s'ajoute aux deux boucles précédentes la boucle d'apprentissage sur les n composantes des n classificateurs c_i. Trois choix sont alors possibles pour insérer la boucle sur les composantes au sein des deux boucles existantes. Nous discuterons des trois algorithmes d'induction obtenus avant de justifier celui que nous retenons.

La plus grande difficulté est l'ambiguïté des exemples non annotés. En effet dans le cas d'une annotation partielle, on ne dispose pas de tous les exemples positifs. Certains d'entre eux font parti des exemples non annotés. La technique de construction de l'ensemble des exemples employée dans le chapitre 4 pour apprendre chaque classificateur c_i n'est plus applicable car elle suppose une annotation complète des documents d'apprentissage. Nous verrons comment résoudre ce problème.

Le plan de ce chapitre est le suivant. La section 5.2 décrit le cadre interactif dans lequel nous nous plaçons en définissant :
- ce qu'est une annotation partielle ;
- comment l'utilisateur est modélisé, *i.e.* les actions qu'il peut réaliser ;
- le scénario interactif retenu.

Ensuite la section 5.3 discute de l'ordre des trois boucles évoquées auparavant, à savoir la boucle de parcours des documents, la boucle d'interactions sur le document courant et la boucle sur les composantes. Cette section présente également notre algorithme d'induction interactif. La section 5.4 montre comment l'apprentissage des classificateurs c_i est réalisé et notamment comment la construction de l'ensemble d'apprentissage, et particulièrement des exemples négatifs, est assurée dans le cas d'une annotation partielle. Enfin la section 5.5 propose l'évaluation expérimentale de notre algorithme interactif d'induction d'extracteurs n-aires. Nos différentes simulations montrent qu'il est possible d'obtenir avec un nombre d'interactions faible des extracteurs aussi performants qu'à partir de documents complètement étiquetés.

5.2 Cadre de travail

La description du cadre interactif passe par la spécification :
- de la notion d'annotation partielle ;
- de la modélisation de l'utilisateur ;
- du scénario interactif envisagé.

5.2.1 Annotation partielle

L'annotation partielle d'un document est définie par un couple d'ensembles : celui des exemples positifs, noté d_p^+, et des négatifs, noté d_p^-, tous les deux étant exprimés en extension. L'annotation partielle du document d est notée $a_p(d)$ avec $a_p(d) = (d_p^+, d_p^-)$.

Une annotation partielle devient complète lorsque l'énumération des exemples positifs est exhaustive, *i.e.* quand tous les positifs sont connus. La donnée des négatifs

n'est alors plus nécessaire et devient implicite.

5.2.2 Modélisation de l'utilisateur

Cette section décrit la modélisation de l'utilisateur dans notre cadre interactif en précisant :
- ce que l'on suppose de lui ;
- les actions qu'il peut réaliser.

5.2.2.1 Hypothèses faites sur l'utilisateur

On suppose que l'utilisateur U est capable de :
- de dire si le résultat de l'application de l'extracteur sur un document est correct ou pas ;
- de fournir des exemples de tuples à extraire et aussi des contre-exemples.

Soit dc un document et w un extracteur. On note $w(dc)$ l'application de l'extracteur w au document dc et $a(dc)$ l'annotation complète du document dc. La première hypothèse suppose que l'utilisateur U peut répondre par oui ou par non à la question : "est-ce que $w(dc)$ est égal à $a(dc)$?". Autrement dit on suppose que $a(dc)$ est implicitement connu de l'utilisateur et qu'il peut comparer l'annotation complète cible $a(dc)$ et $w(dc)$, l'annotation complète réalisée par l'extracteur w.

La seconde hypothèse suppose que U a la capacité de désigner dans un document dc des exemples de tuples à extraire ainsi que des contre-exemples, *i.e.* des tuples à ne pas extraire.

La fonction $extracteurCorrect_U$ rend compte de ces deux hypothèses et modélise les capacités qu'elles supposent. Elle prend en entrée un extracteur w et un document dc et renvoie en sortie :
- la valeur *vrai* si $w(dc)$ est égal à $a(dc)$;
- un ensemble E constitué d'exemples et/ou de contre-exemples sinon.

Plusieurs choix sont possibles pour construire E. Ils rendent compte des différentes manières de corriger les erreurs commises par l'extracteur w. Le choix que nous avons fait est décrit dans la section 5.5 lorsque nous précisons comment l'utilisateur est simulé dans nos expériences. La connaissance de ce choix n'est pas nécessaire à la compréhension de ce qui suit.

5.2.2.2 Actions possibles pour l'utilisateur

Les actions possibles pour l'utilisateur sont les suivantes :
- arrêter le processus d'apprentissage ;
- choisir le document courant.

La première action est modélisée par la fonction $arreterApprentissage_U$ qui renvoie *vrai* si l'utilisateur U décide, selon un certain critère, d'arrêter l'apprentissage et *faux* sinon.

La seconde action permet à l'utilisateur de choisir le document courant. Elle est modélisée par la fonction $choisirDocument_U$ qui renvoie le document courant choisi.

Algorithme 6 Scénario interactif

Entrée: l'utilisateur U ; l'algorithme d'apprentissage $InduireExtracteur$

Notation: D est l'ensemble des documents complètement étiquetés ; dc est le document courant

1: $D = \emptyset$
2: $w =$ `null`
3: **tant que non** $arreterApprentissage_U()$ **faire**
4: $dc = choisirDocument_U()$
5: $a_p(dc) = \emptyset$ # annotation partielle
6: **tant que non** $extracteurCorrect_U(w, dc)$ **faire**
7: soit E les annotations renvoyées par $extracteurCorrect_U(w, dc)$
8: $a_p(dc) = a_p(dc) \cup E$
9: $w = InduireExtracteur(D, dc, a_p(dc))$
10: **fin tant que**
11: $D = D \cup \{dc\}$
12: **fin tant que**

Sortie: w

À nouveau plusieurs possibilités sont envisageables pour le critère d'arrêt de l'apprentissage et pour celui du choix du document courant. Ces points seront précisés dans la section 5.5.

5.2.3 Scénario interactif

Le principe de base de notre scénario interactif est simple : tant que l'extracteur hypothèse w, obtenu à partir d'une annotation partielle, ne satisfait pas l'utilisateur U, ce dernier complète son annotation et relance l'apprentissage jusqu'à l'obtention d'une hypothèse satisfaisante. Ce processus se décline document par document : les annotations de l'utilisateur portent sur le document courant dc qui est partiellement annoté. Un extracteur hypothèse est satisfaisant si lorsqu'il est appliqué à dc, les informations qui en sont extraites sont exactement celles attendues par l'utilisateur U. Si tel est le cas, alors dc devient un document complètement annoté. L'induction d'un extracteur hypothèse s'effectue à partir d'un ensemble de documents complètement annotés, noté D, et à partir du document courant qui est partiellement annoté et qui sert à valider l'hypothèse courante.

Supposons que l'on dispose d'un algorithme d'induction fonctionnant à partir d'un ensemble de documents annotés complètement et à partir d'un document partiellement annoté et notons le $InduireExtracteur$. Le fonctionnement détaillé de scénario interactif considéré dans ce chapitre est exprimé par l'algorithme 6.

L'algorithme 6 est constitué de deux boucles :

– une boucle externe sur les documents choisis par l'utilisateur (ligne 3) ;
– une boucle interne sur le document courant (ligne 6).

La première étape de la boucle interne est le choix d'un document courant dc

par U. Ensuite la boucle interne assure l'induction de l'extracteur hypothèse w : tant que l'annotation de dc par w est différente de l'annotation correcte de dc (que l'on suppose connue de l'utilisateur) l'apprentissage continue. L'induction de w comporte deux étapes :

- à la ligne 8, par le biais de la fonction $extracteurCorrect_U$ et selon les erreurs commises par w, U fournit des exemples annotés (positif et/ou négatif) de dc, notés E, qui viennent s'ajouter à ceux fournis auparavant (stockés dans $a_p(dc)$ qui représente l'annotation partielle de dc) ;
- à la ligne 9 l'algorithme $InduireExtracteur$ est appliqué à D et dc (dont l'annotation partielle est donnée par $a_p(dc)$) pour produire en sortie l'extracteur hypothèse w.

L'induction s'arrête lorsque l'hypothèse w annote complètement dc comme U l'attend. dc est alors ajouté à l'ensemble des documents complètement étiquetés D (ligne 11).

5.3 Algorithmes interactifs d'induction

La problématique de ce chapitre est d'induire un extracteur le moins d'annotations possibles de la part de l'utilisateur. Dans cette section nous présentons comment l'approche de l'algorithme d'induction 5 du chapitre 4 est insérée dans le cadre interactif présenté dans la section précédente (section 5.2).

Rapellons que ce cadre interactif comporte deux boucles :

- l'une permettant de parcourir les documents d'apprentissage proposés par l'utilisateur ;
- et l'autre permettant à l'utilisateur d'interagir sur le document courant avec l'algorithme d'induction.

Rappelons également que l'algorithme d'induction 5 est fondé sur une boucle d'apprentissage sur les composantes des n-uplets à extraire. Un extracteur w est une séquence de n classificateurs c_i notée $w = (c_1, \ldots, c_n)$ (on note aussi $\forall i \in \{1, \ldots, n\}\ w_i = (c_1, \ldots, c_i)$). Chaque classificateur c_i est appris lors de l'étape i de la boucle sur les composantes.

Intégrer cette approche dans notre cadre interactif revient à insérer la boucle sur les composantes entre les deux boucles de l'algorithme 6. Pour cela, il y a trois possibilités, chacune d'entre elles conduisant à un algorithme interactif d'induction différent. La plus simple est de remplacer l'algorithme $InduireExtracteur$ d'induction d'extracteurs de l'algorithme 6 par la boucle d'apprentissage sur les n composantes. Dans ce cas l'utilisateur annote des n-uplets lors de ses interactions sur le document courant. Avec les deux autres alternatives nous verrons que l'utilisateur annote des tuples incomplets de taille i.

Un point commun aux trois algorithmes interactifs obtenus est que dorénavant l'apprentissage d'un classificateur c_i se fait à partir de l'ensemble D de documents complètement annotés et du document courant dc qui est partiellement annoté. Pour l'instant supposons que nous disposons d'une fonction $ApprendreClassificateur$ capable réaliser cette tâche d'apprentissage. La description précise de $ApprendreClassificateur$ est l'objet de la section 5.4). Les trois sections suivantes examinent respectivement les trois algorithmes interactifs d'induction.

5.3.1 Algorithme interactif version 1

La première possibilité pour insérer la boucle sur les composantes est de la placer à l'intérieur de la boucle d'interactions sur le document courant. Ainsi, l'ordre des trois boucles est le suivant :

- la boucle de parcours des documents d'apprentissage ;
- la boucle d'interactions sur le document courant ;
- la boucle sur les composantes.

On obtient l'algorithme 7 dans lequel l'utilisateur annote directement des n-uplets. L'annotation partielle du document courant dc est notée $a_p^n(dc)$. Les interactions ont lieux sur le document courant tant que l'extracteur de n-uplets w_n n'est pas correct (ligne 6). Il n'y a aucune interaction au cours de la boucle sur les composantes (ligne 9). L'apprentissage des classificateurs c_i a lieu ligne 10 à l'aide de la fonction $ApprendreClassificateur$. Elle prend en paramètres l'indice de la composante i, l'ensemble D des documents complètement annotés, le document courant dc et son annotation partielle sur les i-premières composantes, notée $a_p^i(dc)$. $a_p^i(dc)$ est calculée à partir de $a_p^n(dc)$ par projection de l'annotation des n-uplets sur les i premières composantes.

Algorithme 7 Algorithme d'induction interactif, version 1

Entrée: l'utilisateur U

Notation: $a_p^n(dc)$ est l'annotation partielle du document courant dc pour des n-uplets, $a_p^i(dc)$ est l'annotation partielle de dc pour des tuples incomplets de taille i

1: $D = \emptyset$
2: $w_n =$ `null`
3: **tant que non** $arreterApprentissage_U()$ **faire**
4: $dc = choisirDocument_U()$
5: $a_p^n(dc) = \emptyset$
6: **tant que non** $extracteurCorrect_U(w_n, dc)$ **faire**
7: soit E les annotations renvoyées par $extracteurCorrect_U(w_n, dc)$
8: $a_p^n(dc) = a_p^n(dc) \cup E$
9: **pour** $i = 1$ **à** n **faire**
10: $c_i = ApprendreClassificateur(i, D, dc, a_p^i(dc))$
11: **fin pour**
12: **fin tant que**
13: $D = D \cup \{dc\}$
14: **fin tant que**

Sortie: $w_n = (c_1, \ldots, c_n)$

Le calcul de $a_p^i(dc)$ à partir de $a_p^n(dc)$ est simple dans le cas des exemples positifs, *i.e.* des n-uplets à extraire, annotés par l'utilisateur U. La projection d'un n-uplet à extraire sur les i premières composantes donne un tuple incomplet de taille i qui est encore un tuple positif. Par contre dans le cas des exemples négatifs, *i.e.* des n-uplets

à ne pas extraire ou contre-exemples, ce procédé de calcul pose problème. En effet la projection d'un n-uplet sur les i premières composantes ne donne pas nécessairement un tuple de taille i négatif. Considérons par exemple que U annote comme négatif un n-uplet dont les $n-1$ premières composantes sont correctes et seule la composante n est fausse. La projection sur les i premières composantes, avec $i < n$, donne un exemple positif. C'est là le défaut de l'algorithme 7.

Pour remédier à ce défaut, une solution consiste à demander à l'utilisateur d'indiquer quelles sont les composantes correctes d'un n-uplet annoté comme un contre-exemple. Cependant nous considérons que ce type d'interaction est complexe pour un utilisateur final. De plus l'étude de l'organisation des données du chapitre 3 a montré que la disposition des tuples dans les documents semi-structurés peut-être complexe. Ainsi désigner des n-uplets en entier est fastidieux et délicat, notamment lorsque des composantes sont factorisées. Dans ce cas il est plus simple pour l'utilisateur de désigner d'abord les composantes factorisées plutôt que tous les développements des n-uplets.

Pour faciliter la tâche d'annotation de l'utilisateur et conserver des interactions simples, nous proposons deux autres algorithmes interactifs dans lesquels l'annotation interactive est réalisée composante par composante. Dans l'algorithme 8, les annotations sont déclinées d'abord composante par composante puis document par document, tandis que dans l'algorithme 9 c'est l'inverse.

5.3.2 Algorithme interactif version 2

Dans l'algorithme 8, l'ordre des boucles est le suivant :

- la boucle sur les composantes ;
- la boucle de parcours des documents d'apprentissage ;
- la boucle d'interactions sur le document courant.

Ainsi les interactions se déclinent composante par composante.

Pour chaque composante (boucle externe de la ligne 1), l'apprentissage du classificateur c_i continue (boucle de la ligne 4) tant que l'utilisateur U le décide. Il s'agit ici d'une légère variation par rapport aux hypothèses faites sur l'utilisateur dans la section 5.2.2 de la page 111. Ici on suppose aussi que l'utilisateur peut sait reconnaître des tuples incomplets corrects. Cet apprentissage se fait de manière interactive sur le document courant dc, que U choisit à la ligne 5. Il se termine lorsque l'extracteur hypothèse $w_i = (c_1, \ldots, c_i)$ extrait parfaitement les tuples incomplets de taille i de dc (critère d'arrêt de la boucle de la ligne 7).

À chaque étape de la boucle externe sur les composantes, les interactions portent sur la correction de l'extracteur hypothèse w_i. L'inconvénient de cet algorithme est que le passage de la composante i à la composante $i+1$ ne permet plus de remettre en cause l'extracteur hypothèse w_i, ou plus exactement le classificateur c_i. Ce dernier pourrait très bien commettre des erreurs sur le document courant choisi pour l'apprentissage de c_{i+1}. C'est pour remédier à cette limitation que nous proposons l'algorithme 9 dans la section suivante.

Algorithme 8 Algorithme d'induction interactif, version 2

Entrée: l'utilisateur U

Notation: $a_p^i(dc)$ est l'annotation partielle de dc pour des tuples incomplets de taille i

pour $i = 1$ **à** n **faire**
  $D = \emptyset$
  $c_i =$ `null`
  tant que non $arreterApprentissage_U()$ **faire**
    $dc = choisirDocument_U()$
    $a_p^i(dc) = \emptyset$
    tant que non $extracteurCorrect_U(w_i, dc)$ **faire**
      soit E les annotations renvoyées par $extracteurCorrect_U(w_i, dc)$
      $a_p^i(dc) = a_p^i(dc) \cup E$
      $c_i = ApprendreClassificateur(i, D, dc, a_p^i(dc))$
    fin tant que
    $D = D \cup \{dc\}$
  fin tant que
fin pour

Sortie: $w_n = (c_1, \ldots, c_n)$

5.3.3 Algorithme interactif version 3

L'algorithme 9 est l'algorithme interactif que nous retenons dans cette thèse. Par rapport à l'algorithme 8, seule l'imbrication des trois boucles le composant change. Leur ordre est le suivant :

- la boucle de parcours des documents d'apprentissage ;
- la boucle sur les composantes ;
- la boucle d'interactions sur le document courant.

Avec cet ordre des boucles, les interactions se font sur le document courant dc composante par composante tant que l'utilisateur décide de poursuivre l'apprentissage. Le passage à un nouveau document se fait lorsque pour toute composante i l'extracteur hypothèse est correct sur dc.

Cet algorithme ne présente pas le défaut majeur de l'algorithme 8. N'importe lequel des classificateurs c_i peut être remis en cause à la rencontre d'un document courant sur lequel il ne serait pas parfait.

La suite de ce chapitre se focalise d'abord sur la fonction $ApprendreClassificateur$ puis sur l'évaluation expérimentale de l'algorithme 9.

5.4 Apprentissage d'un classificateur c_i

Cette section présente la fonction $ApprendreClassificateur$ qui réalise l'apprentissage d'un classificateur c_i à partir de documents complètement annotés et d'un do-

Algorithme 9 Algorithme d'induction interactif, version 3

Entrée: l'utilisateur U

Notation: $a_p^i(dc)$ est l'annotation partielle de dc pour des tuples incomplets de taille i

1: $D = \emptyset$
2: $c_i = \texttt{null}\ \forall i \in \{1, \ldots, n\}$
3: **tant que non** $arreterApprentissage_U()$ **faire**
4: $dc = choisirDocument_U()$
5: **pour** $i = 1$ **à** n **faire**
6: $a_p^i(dc) = \emptyset$
7: **tant que non** $extracteurCorrect_U(w_i, dc)$ **faire**
8: soit E les annotations renvoyées par $extracteurCorrect_U(w_i, dc)$
9: $a_p^i(dc) = a_p^i(dc) \cup E$
10: $c_i = ApprendreClassificateur(i, D, dc, a_p^i(dc))$
11: **fin tant que**
12: **fin pour**
13: $D = D \cup \{dc\}$
14: **fin tant que**

Sortie: $w_n = (c_1, \ldots, c_n)$

cument partiellement annoté. Bien que les principes de l'algorithme d'induction 5 du chapitre précédent soient repris ici, ils doivent être adaptés pour prendre en compte l'incomplétude de l'annotation du document courant, notamment en ce qui concerne la construction de l'ensemble d'apprentissage du classificateur, et notamment dans le cas exemples négatifs. En effet, l'algorithme d'induction 5 suppose que les documents d'apprentissage sont tous complètement annotés. Ainsi on dispose de tous les exemples positifs et le calcul des exemples négatifs est simple. C'est pour cela qu'avant de détailler la fonction *ApprendreClassificateur* il est nécessaire de préciser comment les exemples négatifs sont obtenus.

5.4.1 Fonction *ApprendreClassificateur*

La fonction *ApprendreClassificateur* est la brique de base pour l'apprentissage des classificateurs c_i à partir de documents complètement annotés et d'un document partiellement annoté. Cette fonction intervient dans les trois algorithmes d'induction interactive décrits dans les trois sections précédentes.

La fonction *ApprendreClassificateur* est décrite par l'algorithme 10. Elle prend en entrée l'indice i de la $i^{\text{ième}}$ composante, l'ensemble D des documents complètement annotés, dc le document courant et $a_p^i(dc)$ son annotation partielle. Elle renvoie en sortie le classificateur c_i.

L'apprentissage de c_i nécessite :

– le calcul des exemples (des tuples de taille i) positifs et négatifs ;

Algorithme 10 Fonction *ApprendreClassificateur* d'apprentissage d'un classificateur c_i à partir des documents complètement annotés D et du document courant dc partiellement annoté

Entrée: i la composante en cours de traitement, l'ensemble D des documents complètement annotés, dc le document courant et $a_p^i(dc)$ son annotation partielle

1: # calculs des exemples positifs et négatifs pour D
2: $S_i^+ = (\bigcup_{d \in D} \{t_i \in S_i^+(d) \mid t_i[i] \neq \texttt{null}\})$
3: $S_i^- = (\bigcup_{d \in D} \{t_i = (t'_{i-1}, l) \mid l \in L(d), t_{i-1} \in S_{i-1}^+, t'_i \notin S_i^+\})$
4: # calculs des exemples positifs et négatifs pour dc
5: $S_i^+ = S_i^+ \bigcup S_i^+(a_p^i(dc))$
6: $S_i^- = S_i^- \bigcup \mathsf{Negp}(i, D, dc, a(dc)) \bigcup S_i^-(a_p^i(dc))$
7: $c_i = A(\bigcup_{x \in S_i^+} \{rep_i(x)\}, \bigcup_{x \in S_i^-} \{rep_i(x)\})$

Sortie: le classificateur c_i

– l'apprentissage proprement dit à partir des représentations des exemples.

Pour les documents complètement annotés de l'ensemble D, les calculs des exemples positifs et négatifs s'effectuent comme dans l'algorithme 5 (lignes 2 et 3). Il en est de même pour le calcul des exemples positifs tirés de l'annotation partielle du document courant dc (ligne 5). Seul diffère le calcul des négatifs à partir de dc. Ceux-ci sont obtenus par la fonction Negp décrite dans la section suivante.

Une fois l'ensemble d'apprentissage constitué, le classificateur c_i est appris, ligne 7, à partir de la représentation des exemples (selon le codage décrit dans la section 4.5 du chapitre 4).

5.4.2 Sélection des exemples négatifs pour le document courant

Avec une annotation partielle, très peu d'exemples sont étiquetés et le nombre d'exemples négatifs est d'autant plus faible qu'il paraît naturel à un utilisateur d'étiqueter prioritairement les exemples positifs. Il est donc nécessaire de découvrir le plus possible d'exemples négatifs à partir de l'annotation partielle, mais aussi de ne pas sélectionner un exemple positif non annoté comme exemple négatif.

En effet, considérons le problème d'extraction n-aire depuis le document de la figure 5.4.2. La relation cible est (*season*,*club*,*score*). Les tuples à extraire sont (2002,PSG,17) et (2002,OM,31). Supposons que lors de la seconde étape de l'apprentissage, *i.e.* $i = 2$, l'utilisateur annote le couple (2002,PSG) comme un exemple positif. En appliquant telle quelle la technique de sélection des exemples négatifs de l'algorithme 5, on obtient les couples suivants : (2002,Season), (2002,2002), (2002,Club), (2002,Score), (2002,17), (2002,OM), et (2002,31). Ainsi le couple (2002,OM) fait partie des exemples négatifs alors qu'il s'agit d'un exemple positif non annoté par l'utilisateur. Une telle inconsistance de l'ensemble d'apprentissage ne peut conduire qu'à un classificateur spécifique (*i.e.* un cas d'apprentissage par cœur ou de sur-apprentissage) ou erroné.

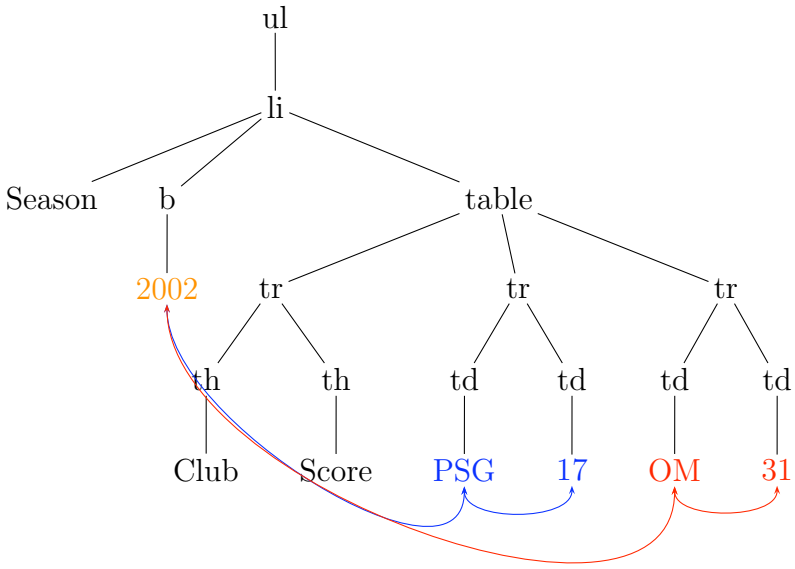


FIG. 5.1 – Organisation avec factorisation dont la relation cible est (*season*,*club*,*score*). La composante factorisée est *season* et les tuples à extraire sont (2002,PSG,17) et (2002,OM,31).

Pour éviter de tels écueils nous proposons d'adapter le procédé de sélection des exemples négatifs de l'algorithme 5 Cette approche est basée sur la notion de *chemin étendu* et de leur généralisation.

5.4.2.1 Notion de chemin étendu

Définition 5.1 (Chemin étendu) *Un chemin étendu est une séquence de couples (label du nœud,position dans la fratrie).*

Par exemple, si l'on considère la feuille contenant la valeur 3 de l'arbre de la figure 5.4.2.1, son chemin étendu est $[(table, 1), (tr, 2), (td, 1), (3, 1)]$.

Le chemin étendu d'un nœud est renvoyé par la fonction *chemin*.

5.4.2.2 Moindre généralisé de chemins étendus

Le moindre généralisé de deux chemins étendus est défini pour des chemins de même taille uniquement. C'est un chemin étendu pouvant comporter plusieurs occurrences du symbole $*$ qui représente soit n'importe quel label soit n'importe quelle position selon son contexte d'utilisation. Il se calcule itérativement en parcourant en parallèle les deux chemins à généraliser et en généralisant à la fois les labels et les positions.

Définition 5.2 (Moindre généralisé de deux chemins étendus) *Soit deux chemins étendus* $c_1 = [(l_{c_1,1}, p_{c_1,1}), \ldots, (l_{c_1,k}, p_{c_1,k})]$ *et*
$c_2 = [(l_{c_2,1}, p_{c_2,1}), \ldots, (l_{c_2,k}, p_{c_2,k})]$ *de même taille* k.

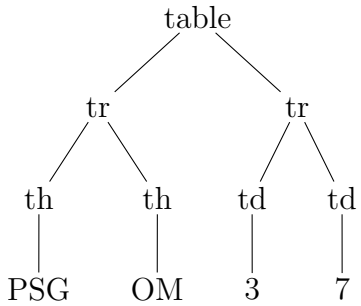


FIG. 5.2 – Une portion d'arbre `HTML`

Le moindre généralisé de c_1 *et de* c_2*, noté* $gce(c_1, c_2) = [(l_{gce,1}, p_{gce,1}), \ldots, (l_{gce,k}, p_{gce,k})]$*, est défini par :*

$$\forall i \in \{1, \ldots, k\}\ l_{gce,i} = \begin{cases} l_{c_1,i} & si\ l_{c_1,i} == l_{c_2,i} \\ * & sinon \end{cases}$$

et par

$$\forall i \in \{1, \ldots, k\}\ p_{gce,i} = \begin{cases} p_{c_1,i} & si\ p_{c_1,i} == p_{c_2,i} \\ * & sinon \end{cases}$$

Par exemple, si l'on considère les feuilles contenant, respectivement, les valeurs PSG et 3 de l'arbre de la figure 5.4.2.1 leurs chemins étendus sont respectivement $[(table, 1), (tr, 1), (th, 1), (PSG, 1)]$. et $[(table, 1), (tr, 2), (td, 1), (3, 1)]$. Leur moindre généralisé est $[(table, 1), (tr, *), (*, 1), (*, 1)]$.

La définition 5.2 s'applique aussi pour généraliser un moindre généralisé et un chemin étendu de même taille entre eux. Elle permet de définir l'opérateur *gce* de calcul du moindre généralisé de deux chemins étendus ou d'un chemin étendu et d'un moindre généralisé.

Définition 5.3 (Couverture d'un chemin étendu par un moindre généralisé) *Soit un chemin étendu* $c = [(l_{c,1}, p_{c,1}), \ldots, (l_{c,k}, p_{c,k})]$ *et un moindre généralisé* $g = [(l_{g,1}, p_{g,1}), \ldots, (l_{g,k}, p_{g,k})]$ *de même taille* k*.*

g couvre c, noté $g \succ c$ *, si et seulement si :*

$$\forall i \in \{1, \ldots, k\}\ (l_{g,i} == * \vee l_{g,i} == l_{c,i}) \wedge (p_{g,i} == * \vee p_{g,i} == p_{c,i})$$

On peut étendre le calcul du moindre généralisé à un ensemble de chemins étendus. C'est le propos de l'algorithme 11, noté GCE, qui calcule la généralisation de l'ensemble de chemins étendus C selon leur taille. Il s'agit d'un algorithme de généralisation par couverture constitué de deux boucles :

– la boucle externe parcourt l'ensemble C tant qu'il n'est pas vide ;

Algorithme 11 Algorithme GCE de généralisation d'un ensemble de chemins étendus

Entrée: un ensemble de chemins C

```
1: G = ∅
2: tant que C ≠ ∅ faire
3:    g = C[0]
4:    C = C − {g}
5:    pour c dans C faire
6:       si taille(c) == taille(g) alors
7:          C = C − {c}
8:          g = gce(g, c)
9:       fin si
10:   fin pour
11:   G = G ∪ {g}
12: fin tant que
```

Sortie: G

– la boucle interne calcule le moindre généralisé de tous les chemins de même taille à l'aide de l'opérateur *gce*.

Les chemins (de même taille) généralisés entre eux sont retirés de C, ce qui assure que cet ensemble se vide. La sortie de l'algorithme est un ensemble de moindres généralisés de chemins étendus, à raison d'un moindre généralisé pour chaque taille de chemins dans C.

5.4.2.3 Fonction Negp

Le fonctionnement de la fonction Negp est inspirée du procédé de sélection des exemples négatifs de l'algorithme 5 du chapitre 4. Nous le rappelons brièvement. Dans l'algorithme 5 les exemples négatifs de l'étape i sont obtenus en complétant les exemples positifs de l'étape $i-1$ par une feuille pour obtenir des tuples de taille i. L'ensemble des feuilles candidates est l'ensemble des feuilles des arbres des documents.

Comme on l'a déjà vu, adoptée la même technique n'est pas toujours possible avec un document partiellement annoté. Le calcul de l'ensemble des feuilles candidates pour compléter des tuples de taille $i-1$ doit être modifié dans deux cas :

– à la première étape d'apprentissage ($i = 1$) ;
– lorsque les $i-1$ premières composantes d'un tuple de taille i sont factorisées.

Il est nécessaire d'écarter les feuilles qui sont susceptibles de produire un exemple positif non étiqueté. Pour ce faire nous proposons de ne conserver que les feuilles dont le chemin étendu n'est pas couvert par le moindre généralisé des chemins menant aux feuilles correspondant à la composante i des tuples positifs étiquetés (qu'ils proviennent de documents complètement ou partiellement annotés). Autrement dit le processus de sélection de la fonction Negp écarte tout exemple dont le chemin étendu de la dernière composante est susceptible d'être celui d'un positif. Dans les autres cas, la sélection

des exemples négatifs pour le document partiellement annoté s'effectue comme dans l'algorithme 5.

La fonction Negp est décrite par l'algorithme 12. Elle prend en entrée l'indice d'étape i, l'ensemble de documents complètement étiquetés D, le document courant dc et son annotation partielle $a_p^i(dc)$. L'ensemble L des feuilles de dc et des documents de D est constitué à la ligne 1. Ensuite si $i = 1$ ou si les $i-1$ premières composantes sont factorisées, alors :

- l'ensemble L_i^+ des feuilles qui sont la composante i des exemples positifs de D et de dc est calculé (ligne 3) ;
- l'ensemble LGG des moindres généralisés des chemins étendus de L_i^+ est calculé par l'application de l'algorithme 11 (ligne 4) ;
- à la ligne 5 seules les feuilles dont le chemin étendu n'est pas couvert par un élément de LGG sont retenues pour former l'ensemble des feuilles candidates C (pour calculer les exemples négatifs).

Sinon C est l'ensemble des feuilles L. Enfin, à la dernière ligne de l'algorithme, s'effectue la sélection des exemples négatifs tirés de dc.

Algorithme 12 Fonction Negp de sélection des exemples négatifs pour le document courant

Entrée: l'indice d'étape i, l'ensemble de documents complètement étiquetés D, le document courant dc et son annotation partielle $a_p^i(dc)$

1: $L = \bigcup_{d \in D} \{l \mid l \in L(d)\} \cup \{l \mid l \in L(dc)\}$
2: **si** $i == 1$ ou si les $i-1$ premières composantes sont factorisées **alors**
3: $\quad L_i^+ = \bigcup_{d \in D} \{l = t_i[i] \mid t_i \in S_i^+(d)\} \cup \{l = t_i[i] \mid t_i \in S_i^+(a_p^i(dc))\}$
4: $\quad LGG = \text{GCE}(\{chemin(l) \mid l \in L_i^+\})$
5: $\quad C = \{l \mid l \in L \wedge \nexists\ g_{taille(l)} \in LGG \mid g_{taille(l)} \succ chemin(l)\}$
6: **sinon**
7: $\quad C = L$
8: **fin si**
9: $S_i^-(\text{dc}) = \{t_i = (t'_{i-1}, l) \mid l \in C, t_{i-1} \in S_{i-1}^+(dc), t'_i \notin S_i^+(a_p^i(dc))\})$

Sortie: $S_i^-(\text{dc})$

5.5 Expériences

Cette section présente l'évaluation expérimentale de l'algorithme 9 d'induction interactif. Pour ce faire, nous avons réalisé une simulation de l'utilisateur et de son comportement interactif avec notre algorithme.

5.5.1 Modélisation de l'utilisateur

La modélisation de l'utilisateur dans nos simulations interactives passe par la spécification de :

– la fonction $choisirDocument_U$ qui retourne le prochain document d'apprentissage ;
– la fonction $extracteurCorrect_U$ qui teste la correction de l'extracteur hypothèse.

La fonction $choisirDocument_U$ renvoie un document choisi au hasard parmi les documents du corpus considéré lors d'une expérience.

Pour spécifier la fonction $extracteurCorrect_U$, il faut définir comment l'ensemble des corrections E, qu'elle renvoie lorsque l'extracteur hypothèse est incorrect, est construit. Les erreurs sont corrigées dans l'ordre du document. E contient systématiquement deux exemples. Lors de la première interaction sur un document vierge de toute annotation, la fonction $extracteurCorrect_U$ renvoie deux exemples positifs (si le document contient au moins deux tuples à extraire). Pour les interactions suivantes, elle renvoie un exemple positif non extrait et un exemple négatif extrait par erreur. Si tous les exemples positifs du document courant sont extraits correctement, alors seul un exemple négatif est retourné. De même si aucun exemple négatif n'est extrait par erreur, $extracteurCorrect_U$ renvoie seulement un positif oublié.

5.5.2 Protocole expérimental

Les expériences sont menées sur le corpus CorporateData avec 10 documents.

Le protocole expérimental est le suivant :

– k documents sont tirés au hasard parmi les 10 pour constituer l'ensemble d'apprentissage ;
– la simulation de l'induction interactive de l'extracteur est lancée ;
– l'extracteur obtenu est ensuite évalué sur les $k-1$ documents restants.

k est le nombre de documents retournés par la fonction $choisirDocument_U$, autrement dit c'est le nombre d'appels de cette fonction et le nombre de tours de la boucle externe de l'algorithme 9. Le but des expériences est d'observer la qualité de l'extracteur obtenu et en combien d'interactions pour un nombre de documents fixés. Comme les expériences du chapitre 4 ont montré qu'avec deux documents au plus il était possible d'obtenir un extracteur parfait sur la plupart des jeux de données du corpus CorporateData, k prendra les valeurs 1 et 2 dans cette section. Ce processus est répété 10 fois.

Les performances de l'extracteur sont évaluées à l'aide des mesures de précision, rappel, F-mesure et couverture. Pour évaluer la quantité d'interactions, on compte le nombre d'annotations réalisées dans le cadre interactif, présenté dans ce chapitre, et le nombre d'annotations réalisées avec l'algorithme non interactif, présenté dans le chapitre précédent. Pour cela, on calcule, pour des extracteurs ayant les mêmes performances, le nombre moyen de composantes désignées par l'utilisateur dans le cas d'une annotation complète des documents, noté NI, ainsi que le nombre moyen de composantes désignées par l'utilisateur avec l'algorithme interactif, noté I. Dans le cas interactif, les composantes désignées par l'utilisateur sont à la fois les composantes sélectionnées par l'utilisateur (exemples positifs) et les composantes dé-sélectionnées (exemple négatifs).

Corpus	P	R	F	C	D	NI	I
BBC	100	100	100	100	1	20	8
BEA 1	75.75	85.91	80.01	81.25	2	66.4	18.50
BEA 2	99.55	100.00	99.77	98.75	2	15.6	8.20
BLS	100	100	100	100	1	87	9.80
BLS 2	100	100	100	100	1	10.68	7.80
BLS 3	100	100	100	100	1	144	10
BLS 4	100	100	100	100	1	18	12
BTS	100	100	100	100	1	60	18
EXCITE 1	100	100	100	100	2	25	18.30

TAB. 5.1 – Résultats des expériences interactives sur le corpus CORPORATEDATA

5.5.3 Résultats

Le tableau 5.1 présente les résultats de l'algorithme 9 obtenus sur le corpus CORPORATEDATA. La première colonne est le nom du jeux de données. Les cinq colonnes suivantes sont, respectivement, la précision, le rappel, la F-mesure, la couverture et le nombre de documents d'apprentissage. Les deux dernières colonnes du tableau 5.1 sont les quantités NI et I.

Les résultats expérimentaux montrent que le nombre d'annotations est plus faible avec l'algorithme interactif que sans. Ce nombre est considérablement réduit sur les corpus BLS et BLS 3 dont les documents contiennent beaucoup de tuples. Les extracteurs induits par l'algorithme interactif sont aussi performants que ceux obtenus par l'algorithme non interactif du chapitre précédent.

5.6 Conclusion

Dans ce chapitre nous exposons un algorithme d'induction interactif d'extracteurs n-aire. Le scénario interactif sur lequel il repose est inspiré de SQUIRREL mais étendu au cas n-aire. L'annotation des documents ne se fait pas avant l'induction, mais au cours de celle-ci. Par souci de simplicité et de confort pour l'utilisateur, les annotations se déclinent document par document, autrement dit l'utilisateur annote un document à la fois, le document courant. Un extracteur hypothèse est appris à partir d'une annotation partielle du document courant. L'extracteur induit propose à l'utilisateur une annotation complète du document courant (obtenue par le procédé d'extraction). L'utilisateur corrige les erreurs de l'extracteur hypothèse et relance l'apprentissage jusqu'à l'obtention d'une hypothèse correcte sur le document courant. Le processus d'induction peut ensuite se poursuivre sur un autre document.

Pour conserver l'approche itérative développée dans le chapitre 4, dont nous avons montré l'efficacité expérimentalement, le processus interactif que nous venons de décrire se déroule composante par composante. Ainsi l'apprentissage interactif d'un extracteur se fait selon une boucle sur les n composantes et est ainsi constitué de n phases d'ap-

prentissage interactif, chacune d'elles permettant d'induire un extracteur pour les tuples partiels de taille i. Comme le document courant dc est en cours d'annotation, l'apprentissage d'un classificateur c_i est adapté pour tenir compte de la connaissance partielle des exemples positifs et négatifs. Enfin nos expériences montrent la validité de notre approche interactive qui permet d'obtenir des extracteurs aussi performants que dans le cas non-interactif, mais avec moins d'effort d'annotation de la part de l'utilisateur.

Conclusion

L'objectif de cette thèse était de proposer un algorithme d'induction supervisée d'extracteurs n-aires pour les documents semi-structurés. Rappelons les critères que nous nous étions fixés :

- gestion des valeurs manquantes ;
- aucune hypothèse sur la disposition des tuples dans les documents
- extraction des composantes et construction des n-uplets simultanées, sans aucun pre ou post-traitement ;
- apprentissage à partir de quelques annotations.

Pour y répondre, nous avons proposé le système PAF. Ce dernier comprend deux algorithmes d'induction d'extracteurs n-aires :

- un algorithme fonctionnant à partir d'une annotation complète des documents d'apprentissage ;
- et un autre, interactif, fonctionnant à partir d'une annotation partielle.

Nos expériences ont montré que ces deux algorithmes atteignent les performances des meilleurs systèmes n-aires. Elles mettent aussi en évidence que, même dans le cas des documents complètement annotés, le nombre d'annotations nécessaire pour induire un extracteur performant est faible. Ce nombre est encore plus réduit par notre algorithme interactif. Dans ce cas l'utilisateur fournit quelques annotations qui amorcent l'apprentissage d'un extracteur hypothèse. Une boucle d'interaction permet ensuite à l'utilisateur de corriger les erreurs de l'hypothèse courante et de relancer l'apprentissage jusqu'à l'obtention d'un extracteur satisfaisant.

Les extracteurs induits par nos deux algorithmes d'apprentissage fonctionnent selon le même procédé d'extraction. Les deux principaux aspect de ce dernier sont les suivants :

- il est incrémental : les tuples sont extraits par taille croissante : d'abord les singletons, ensuite les couples, *etc* et ce jusqu'à l'obtention des n-uplets ;
- il utilise une procédure d'enrichissement de la représentation : les tuples de longueur i sont codés avec la connaissance des tuples extraits de longueur $i-1$.

L'extraction des composantes et la construction des n-uplets sont effectuées simultanément. Les valeurs manquantes ne mettent pas en échec ce procédé d'extraction qui est capable de les gérer et d'extraire des n-uplets dont certaines valeurs sont absentes. Ajoutons également, que notre procédé d'extraction reste applicable et efficace même lorsque l'organisation des données dans les documents semi-structurés est complexe.

Ainsi l'idée que l'on peut induire des programmes d'extraction n-aire pour les documents semi-structurés à partir de quelques exemples de tuples à extraire, et ce quelle que soit l'organisation des n-uplets dans les documents, est démontrée comme pertinente et valide par cette thèse.

Cependant de nombreuses extensions de nos travaux sont possibles.

Notre procédé d'extraction s'applique au cas où une valeur à extraire est exactement une feuille d'un arbre `XML`. Nous ne traitons pas le cas où une valeur à extraire est située sur plusieurs feuilles, ou encore celui où la valeur à extraire est à l'intérieur du texte d'une feuille. Une extension simple de notre algorithme permettrait de traiter le premier cas, quand au second on peut espérer le résoudre en composant un extracteur arborescent et un extracteur textuel. On pourrait également employer des transformations sur les documents pour les ramener dans notre cadre de travail, ce qui rendrait nos algorithmes applicables.

Une autre piste d'extension est celle d'intégrer de l'apprentissage actif dans notre algorithme interactif. Un algorithme actif guide son apprentissage en interrogeant l'utilisateur. Les questions peuvent portées sur la correction d'une hypothèse ou l'étiquette d'exemples choisis par l'algorithme lui-même.

Appliquer une telle approche à notre algorithme interactif d'induction consisterait, par exemple, à ce que l'algorithme demande à l'utilisateur d'annoter uniquement les tuples susceptibles de faire progresser l'apprentissage le plus rapidement possible. Ces exemples seraient proposés par l'algorithme d'induction en fonction des exemples déjà annotés et du comportement de l'extracteur hypothèse.

Une autre possibilité concerne le choix du prochain document courant. Actuellement ce choix est réalisé par l'utilisateur, mais il pourrait être choisi par l'algorithme d'induction. La motivation est ici encore de faire progresser le plus rapidement possible en demandant à l'utilisateur d'annoter un document particulier.

Annexe A

Annexes : Quelques systèmes d'induction supervisée d'extracteurs

Cette annexe présente cinq algorithmes d'induction supervisée d'extracteurs n-aires : WIEN [51], SOFT MEALY [42], STALKER [65], LIPX [81, 82] , et SQUIRREL$_n$ [56]. Pour une étude plus détaillée des systèmes d'extraction d'information, nous renvoyons le lecteur à [55]. Les systèmes examinés ici sont tous des systèmes d'extraction n-aire qui s'appliquent aux documents `HTML` et `XML`.

Rappelons qu'un système d'extraction d'information supervisé prend en entrée des documents dans lesquels les informations à extraire sont annotées. Il fournit en sortie l'extracteur induit au cours d'un processus d'apprentissage à partir de l'annotation des documents.

Deux types d'approches se dégagent : les méthodes qui fonctionnent avec des documents complètement étiquetés, et celles fonctionnant avec des annotations partielles. Les premières imposent que tous les n-uplets à extraire soient annotés : c'est le cas de WIEN, SOFT MEALY, LIPX et SQUIRREL$_n$. Les secondes approches sont capables d'induire un extracteur à partir d'un faible nombre d'annotations, par exemple le système STALKER. Cette particularité permet à ces derniers d'être embarquées au sein d'un système interactif, comme l'algorithme SQUIRREL [12], version unaire de SQUIRREL$_n$, qui est distribué comme une extension du navigateur FIREFOX.

Comme nous l'avons vu précédemment, il y a deux manières de représenter les documents :

- comme des séquences (WIEN, SOFT MEALY, STALKER) ;
- comme des arbres (LIPX, SQUIRREL$_n$).

La représentation choisie implique des techniques algorithmiques différentes pour l'extraction et l'induction. Dans WIEN, SOFT MEALY et STALKER la valeur d'une composante est une sous-séquence de la séquence représentant un document, tandis que dans LIPX et SQUIRREL$_n$ la valeur d'une composante est un nœud de l'arbre.

Nous ferons une dernière distinction majeure parmi les approches d'extraction n-aire. D'un côté on trouve des méthodes basées sur n extracteurs unaire suivis d'un

post-processus de re-combinaison pour former des n-uplets à partir des différentes composantes extraites. Ce post-traitement est :

- soit une heuristique qui généralement considère simplement que les n-uplets sont les uns à la suite des autres dans le document ;
- soit à la charge de l'utilisateur. C'est le cas de STALKER qui propose un formalisme arborescent pour décrire l'organisation des tuples [1].

De l'autre il existe des approches assurant l'extraction directe des n-uplets, sans aucun post-traitement. C'est le cas de WIEN, SOFT MEALY, LIPX et SQUIRREL$_n$ qui réalisent simultanément l'extraction des valeurs des composantes et la construction des n-uplets.

Les cinq sections suivantes présentent, respectivement, les algorithmes WIEN, SOFT MEALY, STALKER, LIPX et SQUIRREL$_n$.

A.1 WIEN

WIEN [51, 52, 53] est un système pionnier. Il s'applique aux documents `HTML` qui sont traités comme des chaînes de caractères et qui doivent être complètement annotés. La valeur d'une composante d'un n-uplet est une séquence de caractères. Elle est représentée par ses indices de début b et de fin e dans la séquence de caractères du document. Un n-uplet est ainsi codé par n couples d'indices $((b_1, e_1), \ldots, (b_n, e_n))$.

L'extraction de la valeur d'une composante est réalisée en repérant ses *délimiteurs* gauche et droit. Un délimiteur est une séquence de caractères qui se trouve soit avant la donnée à extraire, dans ce cas on parle de délimiteur gauche et on le note l, soit après elle, il s'agit alors d'un délimiteur droit, noté r. Par exemple, les valeurs `PSG` et `OM` de la composante `Club` du document `HTML` de la figure A.1 ont comme délimiteur gauche la séquence `<td>` et comme délimiteur droit `</td>`.

```
<html> <body> <table>
<tr> <th>Club</th> <th>Score</th> </tr>
<tr> <td>PSG</td> <td>17</td> </tr>
<tr> <td>OM</td> <td>31</td> </tr>
</table> </body> </html>
```

FIG. A.1 – Exemple de document `HTML`

WIEN comporte six classes d'extracteurs à base de délimiteurs. La plus simple d'entre elles est la classe LR. Un extracteur LR utilise un couple de délimiteurs (l, r) pour repérer chaque composante. Il est défini par n paires de délimiteurs $((l_1, r_1), \ldots, (l_n, r_n))$ et par la fonction $extraire_{LR}$, qui est décrite par l'algorithme 13 en pseudo-code. Cette fonction prend en entrée un extracteur $w = ((l_1, r_1), \ldots, (l_n, r_n))$ et un document d. Le rôle de cette fonction est d'appliquer l'extracteur w au document d. L'algorithme d'extraction est séquentiel et composé de deux boucles :

- une boucle externe qui assure la construction des tuples (ligne 2) ;

[1] le système WL^2 propose quand à lui un formalisme graphique pour la même tâche[43]

– une boucle interne sur les composantes du tuple courant, noté t (ligne 4).

La séquence de caractères du document d est parcourue du début à la fin au cours de l'extraction. La boucle externe s'exécute tant qu'il existe une occurrence du délimiteur l_1 dans le document, dont la présence indique l'existence d'un nouveau tuple. La boucle interne porte sur les n composantes de la relation n-aire cible. À chaque étape de cette boucle, la sous-séquence délimitée par (l_i, r_i) est extraite comme $i^{\text{ième}}$ composante du tuple courant t (lignes 5 et 6). Une fois la boucle interne terminée, le n-uplet t est ajouté à l'ensemble T des tuples extraits (à la ligne 8), qui est la sortie de la fonction $extraire_{LR}$.

Algorithme 13 Fonction $extraire_{LR}$

Entrée: un document d vu comme une séquence de caractères, un extrateur LR $w = ((l_1, r_1), \ldots, (l_n, r_n))$

1: $T = \emptyset$
2: **tant que** une autre occurrence de l_1 existe dans d **faire**
3: $t = \emptyset$
4: **pour** $i = 1$ **à** n **faire**
5: soit v_i la sous-séquence de d délimitée par (l_i, r_i)
6: $t = (t, v_i)$
7: **fin pour**
8: $T = T \cup \{t\}$
9: **fin tant que**

Sortie: l'ensemble T des n-uplets extraits

L'induction d'un extracteur LR consiste à trouver les délimiteurs $((l_1, r_1), \ldots, (l_n, r_n))$ à partir d'un ensemble de documents complètement annotés. Sans décrire précisément l'algorithme d'induction, on retiendra qu'il tente de trouver des délimiteurs qui respectent certaines contraintes. Un délimiteur, gauche ou droit, ne doit pas être une sous-chaîne d'une valeur à extraire. Un délimiteur gauche l_i doit être un suffixe commun à toutes les sous-séquences qui se trouvent entre les composantes $i-1$ et i. Un délimiteur droit r_i doit être un préfixe commun à toutes les sous-séquences suivant immédiatement chaque valeur de la composante i des n-uplets de l'ensemble d'apprentissage.

Les cinq autres classes d'extracteurs sont : $HLRT$, $OCLR$, $HOCLRT$, NLR et $NHLRT$. Les extracteurs $HLRT$ utilisent deux délimiteurs supplémentaires h et t qui démarquent la zone du document qui contient les n-uplets. L'extraction des n-uplets commence dès que le délimiteur h est rencontré, lors de la lecture du document, et s'arrête après le délimiteur t. L'extraction des composantes d'un n-uplet s'effectue comme dans la boucle interne de l'algorithme 13. Les extracteurs $OCLR$ font également intervenir deux délimiteurs supplémentaires o et c. Comme avec les extracteurs $HLRT$, c'est à nouveau la boucle externe de l'algorithme 13 qui est modifiée, tandis que la boucle interne reste identique. La rencontre du délimiteur o indique qu'un nouveau tuple est à extraire. La boucle interne d'extraction des composantes du n-uplet courant est appliquée. Puis la lecture du document reprend jusqu'à la rencontre de c. L'extraction

se termine lorsqu'il n'y a plus d'occurrence de o dans le document. La classe $HOCLRT$ combine les caractéristiques des deux classes précédentes. Enfin les classes NLR et $NHLRT$ sont une adaptation des extracteurs LR et $HLRT$ pour traiter un certain type de structures imbriquées. Pour chacune de ces classes, un algorithme d'extraction et un autre d'induction sont proposés.

Outre le fait que les extracteurs proposés par WIEN imposent que les délimiteurs d'une composante soient les mêmes pour tous les n-uplets (une disjonction de délimiteurs n'est pas exprimable), ils possèdent également deux défauts.

D'une part WIEN fait une hypothèse forte sur l'organisation des tuples dans les documents : les tuples sont consécutifs et leur composantes sont obligatoirement dans le même ordre. Cette hypothèse est vérifiée lorsque les données sont dans une table car l'ordre des colonnes est naturellement fixe et identique dans toute la table. Cependant elle ne l'est plus nécessairement lorsque les n-uplets sont dans une liste dans laquelle l'ordre des composantes peut changer d'un tuple à l'autre.

D'autre part, les valeurs manquantes ne sont pas gérées convenablement en extraction. Deux cas de figures sont envisageables. Si seule la valeur est manquante, mais que ses délimiteurs (l_i, r_i) sont présents dans le document [2] alors, conformément à ce que l'on attend, la chaîne de caractères vide est extraite. Par contre si à la fois la valeur et les délimiteurs sont absents [3] un décalage dans les composantes a lieu. Soit i la composante absente. La procédure d'extraction recherche le délimiteur l_i du tuple courant t_k. Mais la prochaine occurrence de ce délimiteur se trouve dans le tuple suivant t_{k+1}, ce qui entraîne un mélange des $i-1$ premières composantes de t_k avec les i dernières de t_{k+1}. Ainsi le tuple t_k extrait est $(v_{k,1}, \ldots, v_{k+1,i}, \ldots, v_{k+1,n})$. Le décalage peut prendre de diverses formes, notamment s'il existe $i \neq j$ tels que $l_i = l_j$. Précisons enfin que l'induction d'un extracteur est tout simplement impossible lorsque les documents d'apprentissage contiennent des tuples dont certaines valeurs sont manquantes.

A.2 SOFTMEALY

On peut voir SOFT MEALY [42] comme une amélioration des extracteurs $HLRT$ de WIEN, afin de remédier aux lacunes de ce dernier : l'ordre dans lequel les valeurs des composantes des n-uplets apparaissent est figé et les valeurs manquantes perturbent la procédure d'extraction et rendent inapplicable l'algorithme d'induction. De grandes similarités sont observables entre WIEN et SOFT MEALY. En effet tous les deux considèrent les documents comme des séquences et induisent des extracteurs qui repèrent les n-uplets à extraire grâce à des motifs textuels.

Dans SOFT MEALY, un document est représenté par sa séquence de tokens. Un token est une suite de caractères définie par une expression régulière. Les différents types de tokens (alphabétique,numérique,ponctuation,*etc*) sont organisés selon une typologie arborescente, les types de bases sont aux feuilles et les plus généraux dans les nœuds rencontrés en remontant vers la racine de l'arbre. Une donnée à extraire est une séquence

[2] c'est par exemple le cas lorsqu'une cellule d'une table est vide

[3] comme c'est le cas dans une liste

de tokens et elle est identifiée par le couple de *séparateurs* (début,fin), noté (d, f), qui la délimitent, d se trouvant juste avant la donnée et f juste après. Un séparateur est une position entre deux tokens consécutifs. Il est représenté par son contexte gauche et droit, c'est à dire par une sous séquence de tokens se trouvant avant et après lui. Par exemple, dans le document de la figure A.1, le séparateur qui se trouve juste avant la valeur `PSG` a comme contexte gauche la séquence de tokens `> < td >` et comme contexte droit `PSG`. Un séparateur est plus expressif qu'un délimiteur. En effet si l'on compare un séparateur de début d à un délimiteur gauche l, alors d exprime comme l quel est le contexte gauche de la donnée à extraire. Par contre d exprime aussi, grâce à son contexte droit, la forme prise par la donnée ou par une de ses parties. La taille des contextes est limitée par la contrainte suivante : les contextes d'un séparateur, début ou fin, d'un donnée à extraire ne doit pas contenir un séparateur d'une autre donnée à extraire.

Le formalisme pour décrire un extracteur est celui des *transducteurs à états finis* [6]. Un transducteur de mots est un automate de mots qui produit un mot de l'alphabet de sortie, en fonction de l'état dans lequel il se trouve et de la lecture du mot d'entrée. L'algorithme d'extraction de SOFT MEALY est conçu dans un esprit similaire à celui des extracteurs $HLRT$ de WIEN. Il fait intervenir deux transducteurs :

- un pour déterminer la zone du document qui contient les n-uplets ;
- et un autre pour extraire les n-uplets de ladite zone.

Le premier transducteur a le même rôle que les délimiteurs h et t d'un extracteur $HLRT$. La sortie de ce transducteur est l'entrée du second, qui est chargé d'extraire les n-uplets. Dans SOFT MEALY, un transducteur prend en entrée la séquence de séparateurs du document, ou d'une de ses parties, et produit en sortie soit un singleton pour le transducteur délimitant la zone des données, soit l'ensemble des n-uplets extraits dans le cas du second transducteur. À partir de l'état initial, la lecture de la séquence de séparateurs d'entrée commence. La transition entre deux états a lieu si la règle de transition associée à l'arrête reliant les deux états est vérifiée. Cette règle impose des contraintes sur les contextes du séparateur courant. Si ses contextes gauche et droit sont conformes à la règle, la transition a lieu et une sortie est produite. Après chaque transition un nouveau séparateur est lu. Le fonctionnement du transducteur s'achève lorsque tous les séparateurs ont été lus.

Le premier des deux transducteurs décrits auparavant est extrêmement simple. Son rôle est d'extraire du document la séquence de tokens, dans laquelle se trouvent les n-uplets, comprise entre deux séparateurs notés h et t. Il est constitué de trois états :

- un état initial qui est quitté lorsque le contexte gauche du séparateur courant correspond à celui de h ;
- un état intermédiaire dans lequel le transducteur reste tant que le contexte droit du séparateur courant n'est pas celui de t
- et un état final dans lequel on arrive quand le transducteur quitte l'état précédant.

Le second transducteur, le transducteur de n-uplets, est plus complexe. Il comporte un état initial et un unique état final. À chaque composante i à extraire correspond deux états : l'état i dans lequel le transducteur restera, grâce à une boucle, pour extraire la séquence de tokens de la composante i et l'état $\overline{i}$, qui correspond à la séquence de tokens à ignorer avant de rencontrer une autre valeur à extraire. Les transitions qui arrivent dans un état i, sauf les boucles, correspondent à la lecture du séparateur début d de la

composante i. Les transitions qui sortent d'un état i, sauf les boucles, correspondent à la lecture du séparateur fin f de la composante i. Les règles des boucles sur les états i assurent l'extraction de la valeur de la composante i en recopiant sur la sortie le token à droite du séparateur courant. Le transducteur de n-uplets est appliqué tant que la séquence de séparateurs en entrée n'est pas épuisée. Chacune de ses applications assure l'extraction d'un n-uplet.

L'induction d'un extracteur se fait à partir d'un ensemble de documents complètement annotés, et consiste à apprendre les deux transducteurs décrits précédemment. Ces apprentissages se font selon des procédés très proches. C'est pourquoi nous décrivons seulement l'algorithme d'apprentissage du transducteur de n-uplets. La topologie de ce transducteur est donnée par les n-uplets de l'ensemble d'apprentissage. À chaque composante i à extraire correspond les deux états i et $\overline{i}$. Il y a une boucle sur chaque état, sauf l'état final, ainsi qu'une transition entre les états i et $\overline{i}$. Les autres transitions sont fixées de telle sorte qu'à chaque n-uplet de l'ensemble d'apprentissage corresponde un chemin dans le transducteur de l'état initial vers l'état final. Ainsi différents ordres sur les composantes sont envisageables. L'apprentissage proprement dit consiste donc à apprendre les règles de transitions entre les états. Les règles des boucles sur les états i consistent à écrire sur la sortie le token à droite du séparateur courant, sans imposer de contrainte sur l'entrée. Les autres transitions, en dehors des boucles, sont celles qui partent ou qui arrivent d'un état i. Cet apprentissage se fait par la généralisation des contextes des séparateurs, de l'ensemble d'apprentissage, associés à l'état. Les contextes sont alignés, à droite pour les contextes gauche et inversement pour les contextes droits. Une contrainte sur la forme des contextes permet de les borner. On obtient ainsi une matrice de séquences de tokens, dont le nombre de lignes est le nombre de contextes et le nombre de colonnes la taille, comptée en tokens, du plus grand contexte. La généralisation procède colonne par colonne en remplaçant les tokens d'une même colonne plaçant par leur plus petit ancêtre commun dans la hiérarchie de tokens. Un contexte généralisé doit couvrir les séparateurs correspondant aux éléments à extraire, mais pas les autres. Ce critère détermine l'arrêt de l'apprentissage.

Les résultats expérimentaux montrent que SOFT MEALY est comparable aux autres systèmes. Cependant, contrairement à WIEN, il tolère la présence de valeurs manquantes, aussi bien en extraction qu'en apprentissage, et différents ordres des composantes. Néanmoins, son expressivité est limitée car, pour un ordre des composantes fixé, un seul contexte généralisé est utilisé pour repérer le début, ou la fin, d'une valeur. Une disjonction de contextes est impossible. Ainsi WIEN et SOFT MEALY souffrent du même défaut en interdisant les disjonctions.

A.3 STALKER

[65] présente STALKER, un système d'extraction n-aire fondé sur n extracteurs unaires. Contrairement à WIEN et SOFT MEALY, les règles d'extractions de STALKER peuvent exprimer des disjonctions.

Un document est vu comme une séquence de tokens. Un token est soit une séquence de caractères alphanumérique, soit une séquence de caractères non alphanumériques, soit une balise `HTML`.

Un extracteur est décrit par un arbre EC, fourni par l'utilisateur, ainsi que par un ensemble de règles d'extraction. L'arbre EC est un formalisme arborescent pour décrire la structure du document et l'organisation des données. Les feuilles de l'arbre EC représentent les composantes à extraire : à chaque composante correspond une feuille. Un nœud interne, à part la racine de l'arbre qui ne possède pas de rôle particulier, représente une liste dont les éléments sont ses fils. L'imbrication des listes est possible. Cet arbre décrit l'organisation logique des données dans le document, mais également un procédé d'extraction hiérarchique. La racine contient à la séquence de tokens S du document dans son intégrité, et chacun de ses fils à une sous-séquence de S. Ainsi à chaque nœud de l'arbre est associée une sous-séquence issue de la séquence contenue dans son père. Une règle d'extraction, à base de délimiteurs, est aussi associée à chaque nœud. Elle permet d'extraire la séquence associée à un nœud de celle de son père, en repérant des délimiteurs. La règle d'un nœud représentant une liste est une règle itérative qui décompose la séquence associée au nœud en éléments individuels. L'algorithme d'extraction procède en parcourant l'arbre et en appliquant à chaque nœud la règle d'extraction. La séquence de tokens extraite est le point de départ de la règle d'extraction suivante.

L'algorithme d'induction consiste à déterminer les règles d'extraction des nœuds de l'arbre EC fourni par l'utilisateur. Les valeurs des n-uplets annotés par l'utilisateur sont reliées aux feuilles adéquates de l'arbre. Cet apprentissage peut se faire à partir de documents partiellement annotés. Il consiste à apprendre les règles d'extraction avec un algorithme spécifique et complexe, qui de plus est assez mal décrit dans [65]. C'est pourquoi nous ne le présentons pas. Nous préciserons seulement que le choix et le raffinement des règles sont guidés par l'utilisation de nombreuses heuristiques (13 au total). De plus les règles apprises subissent encore un post-traitement très spécifique à leur formalisme. Autrement dit, la conception de l'algorithme d'induction de STALKER nous semble trop empirique.

Comme WIEN, STALKER fait une hypothèse sur l'organisation des données dans les documents en supposant que les composantes apparaissent toujours dans le même ordre. Mais le principal défaut de STALKER est de déléguer la procédure de ré-association des valeurs des composantes en n-uplets à l'utilisateur, qui se voit alors chargé de la construction de l'arbre EC fourni à l'algorithme d'induction. Imposer la conception de cette procédure à l'utilisateur est contraire à l'esprit de l'induction d'extracteurs à partir d'exemples annotés.

De plus l'évaluation expérimentale publiée dans [65] est critiquable car les résultats présentés concernent uniquement des tâches d'extraction unaires. Les performances de cette approche sur des tâches n-aires ne sont pas connues. Ajoutons enfin que malgré un protocole expérimental correctement décrit, la mesure utilisée pour juger de la qualité d'un extracteur n'est pas une mesure classique (comme la précision, le rappel ou la F-mesure) et n'est pas clairement définie.

A.4 LIPX

LIPX [81, 82] est un système n-aire qui traite aussi bien les documents non-structurés que les documents arborescents. On se limite ici aux documents `HTML` ou `XML` qui sont vus comme des arbres. La technique d'apprentissage automatique utilisée est la *programmation logique inductive* [64]. Un document arborescent est représenté en logique du premier ordre par un ensemble de prédicats. Il en est de même pour les n-uplets à extraire. L'extracteur est un ensemble de règles d'extraction n-aire, tel que chaque règle est décrite par une clause basée sur les prédicats contraignant le contexte des données à extraire. L'apprentissage est à base de moindres généralisés [4] de clauses du premier ordre [69].

La valeur d'une composante d'un n-uplet est représentée par un *span*, qui lui même est codé par un prédicat particulier. Un span est un triplet (n, l, r) où n est un nœud de l'arbre, l et r sont des indices dans la séquence des fils de n (avec $l < r$). Un span définit le sous-arbre enraciné en n et délimité par sa branche la plus à gauche (indiquée par l) et sa branche la plus à droite (indiquée par r). À chaque portion de texte du document correspond un span dit *minimum* dans le sens où il s'agit du plus petit span incluant tous les éléments du texte.

Un n-uplet à extraire est représenté par une clause dont la tête contient n variables, une pour chaque composante du n-uplet. Le corps de la clause est la conjonction des n prédicats représentant chacun une composante. Les clauses décrivant les n-uplets à extraire sont saturées à partir d'une connaissance du domaine, modélisée par un programme logique, qui représente la description des exemples. On obtient une représentation étendue des n-uplets, constituée d'une vingtaine de prédicats de base [5]. Cette représentation des n-uplets combine à la fois des propriétés de la vue arborescente ainsi que des propriétés de la vue textuelle des documents.

L'induction d'extracteurs, à base de moindres généralisés de clauses, se fait à partir d'un corpus de documents complètement étiquetés, c'est à dire de documents dans lesquels tous les n-uplets à extraire sont annotés. L'algorithme d'apprentissage est un algorithme de couverture séquentiel (voir [62] page 275). Une règle d'extraction est le moindre généralisé de deux clauses, chacune décrivant un n-uplet à extraire. Ce procédé de généralisation continue : une nouvelle règle d'extraction est obtenue en calculant le moindre généralisé de la règle précédente avec un autre exemple, qui n'a participé à aucun calcul jusque là. Une généralisation est acceptée si la règle obtenue extrait uniquement des n-uplets à extraire (pas nécessairement tous) de l'ensemble d'apprentissage. Ce critère impose que les documents soient complètement annotés même si seulement quelques exemples sont nécessaires pour induire l'extracteur cible. Si la règle est acceptée, le procédé continue. Sinon, l'exemple qui a conduit à la règle rejetée est mis à l'écart dans l'ensemble des exemples *non couverts*, c'est à dire l'ensemble des exemples qui n'ont pas participé à la généralisation d'une règle. L'algorithme revient à la règle acceptée lors de l'étape précédente et le processus de généralisation continue. Lorsqu'on ne peut plus ajouter d'exemple à la règle courante sans qu'elle viole le critère précédant, le processus généralisation s'arrête. La règle obtenue est ajoutée à l'ensemble

[4] les moindres généralisés seront abordés plus en détail dans la section 2.2.3.2

[5] chaque prédicat pouvant apparaître plusieurs fois avec des arguments différents

des règles. L'algorithme recommence avec les exemples non couverts, et ce jusqu'à ce que tous ceux de l'ensemble d'apprentissage soient couverts. L'ensemble de règles ainsi produit constitue l'extracteur induit.

Les résultats expérimentaux de [81, 82] sont comparables aux autres approches et meilleurs sur les corpus comportant des valeurs manquantes. Contrairement à WIEN et STALKER, LIPX ne fait d'hypothèse sur l'organisation des données dans les documents. Les n composantes ne respectent pas toujours le même ordre. De plus il gère également les valeurs manquantes. Cependant la principale critique qu'on peut lui adresser est que le coût algorithmique pour déterminer si une règle extrait un n-uplet ou pas est élevé. L'application d'une règle d'extraction est réalisée à l'aide du test de θ-subsomption. Or il est bien connu en programmation logique inductive que ce test est NP-complet. En effet, dans [82] l'auteur rapporte des durées d'exécution des algorithmes d'induction et surtout d'extraction gravement pénalisantes : 30 minutes pour induire un extracteur et 7 minutes pour extraire les données d'un document.

A.5 SQUIRREL$_n$

SQUIRREL$_n$ [56] est un algorithme d'induction d'extracteurs n-aire fondé sur l'inférence grammaticale dans les arbres. Il s'agit d'une amélioration du système SQUIRREL [12, 14] qui ne traite que le cas unaire. Un document `HTML` ou `XML` est vu comme un arbre. L'extraction porte sur des n-uplets de nœuds. Une donnée à extraire est exactement un nœud de l'arbre et la plupart du temps, en pratique, une feuille texte.

Un extracteur est ici un transducteur de sélection de n-uplets. Il s'agit d'un automate d'arbre [20] définissant une requête n-aire, c'est à dire une fonction de sélection de n-uplets dans les arbres. Considérons l'ensemble des arbres définissables sur la signature Σ. Une requête n-aire q est une fonction de l'ensemble des arbres $t \in T_\Sigma$ vers l'ensemble des n-uplets $q(t) \subseteq noeuds(t)^n$, où $noeuds(t)$ désigne l'ensemble des nœuds de t. Une telle requête peut être définie par une formule logique monadique du second ordre. Si la requête est régulière, elle est également définissable par un automate d'arbre sur $\Sigma \times Bool^n$ et réciproquement. Ces liens étroits entre logique et automates d'arbre [80, 20] permettent de transformer l'apprentissage de requêtes n-aire en celui d'automates d'arbre [14, 56]. Toute requête n-aire définissable en logique monadique du second ordre est représentable par un transducteur de sélection de n-uplets.

D'une manière générale, un transducteur est un automate qui à chaque transition produit une sortie en fonction de l'entrée. Un transducteur de sélection de n-uplets est un automate d'arbre déterministe sur $\Sigma \times Bool^n$. Chaque arbre accepté par cet automate contient exactement un n-uplet à extraire. Les composantes peuvent être identifiées car le transducteur transforme le label ce chaque nœud en un vecteur de n booléens, la position d'un 1 codant la composante affectée. Les résultats d'une requête d'extraction pour un arbre t peuvent être calculés de manière efficace par deux passes du transducteur sur t (une ascendante et l'autre descendante).

L'algorithme d'induction d'un extracteur est une variante du célèbre algorithme d'inférence grammaticale RPNI [66] adapté aux arbres. L'apprentissage se fait à partir

Algorithme	WIEN	SOFT MEALY	STALKER	LIPX	SQUIRREL$_n$
Gestion des valeurs manquantes	×	√	√	√	×
Aucune hypothèse sur l'organisation des données	×	√	×	√	√
Extraction n-aire directe	√	√	×	√	√
Apprentissage à partir d'une annotation partielle	×	×	√	×	×

TAB. A.1 – Comparaisons des algorithmes WIEN, SOFT MEALY, STALKER, LIPX et SQUIRREL$_n$

d'exemples complètement étiquetés [6]. La complexité de l'algorithme d'apprentissage est polynomial en temps et en la taille des données d'entrée.

Les performances de SQUIRREL$_n$ sont prometteuses car les premiers résultats expérimentaux sont encourageants. De plus les extracteurs obtenus sont efficaces et utilisables en pratique. Cependant les valeurs manquantes ne sont pas gérer : les n-uplets dont certaines composantes sont absentes ne peuvent être extraits.

A.6 Bilan

Le tableau A.1 dresse un bilan des systèmes d'induction examinés dans les sections précédentes. Les critères d'analyse retenus sont :

- la capacité à gérer les valeurs manquantes ;
- le fait de ne pas faire d'hypothèse sur l'organisation des données dans les documents ;
- l'extraction directe des n-uplets ;
- la possibilité d'apprendre à partir d'une annotation partielle.

Dans le tableau A.1, les symboles √et ×signifient respectivement que l'algorithme valide le critère considéré ou pas.

L'examen de ce tableau nous montre qu'aucun des cinq algorithmes ne valident tous les critères en même temps.

[6] seul SQUIRREL propose un algorithme d'apprentissage à partir de documents partiellement annotés

Bibliographie

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web : from relations to semistructured data and XML*. Morgan Kaufmann, 2000.

[2] Serge Abiteboul. Querying semi-structured data. In *ICDT*, pages 1–18, 1997.

[3] Arvind Arasu and Hector Garcia-Molina. Extracting structured data from web pages. In *Proceedings of international conference on Management of data*, pages 337–348, 2003.

[4] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual web information extraction with lixto. In *28th International Conference on Very Large Data Bases*, pages 119–128, 2001.

[5] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 2001.

[6] J. Berstel. *Transductions and Context-Free Languages*. Teubner Studienbucher, 1979.

[7] P. Bohunsky and W. Gatterbauer. Table extraction using spatial reasoning on the css2 visual box model, 2006.

[8] Eric Brill. A simple rule-based part of speech tagger. In *Proceedings of the third conference on Applied natural language processing*, pages 152–155. Association for Computational Linguistics, 1992.

[9] Peter Buneman. Semistructured data. In *PODS '97 : Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 117–121, New York, NY, USA, 1997. ACM Press.

[10] M. E. Califf, F. Ciravegna, D. Freitag, C. Giuliano, N. Kushmerick, A. Lavelli, and L. Romano. A critical survey of the methodology for ie evaluation. In *Proceedings of LREC 2004*, 2004.

[11] M.E. Califf. Relational learning techniques for natural language information extraction. Technical Report AI98-276, IA Laboratory, university of Texas of Austin, 1998.

[12] Julien Carme. *Inférence de requêtes dans les arbres et applications à l'extraction d'informations sur le Web*. PhD thesis, Université Charles-de-Gaulle - Lille 3, September 2005.

[13] Julien Carme, Rémi Gilleron, Aurélien Lemay, and Joachim Niehren. Interactive learning of node selecting tree transducer. In *IJCAI Workshop on Grammatical Inference*, 2005.

[14] Julien Carme, Rémi Gilleron, Aurélien Lemay, and Joachim Niehren. Interactive learning of node selecting tree transducers. *Machine Learning*, 66(1) :33–67, 2007.

[15] C. Chang and S. Lui. IEPAD : Information extraction based on pattern discovery. In *Proceedings of Int. World Wide Web Conference*, 2001.

[16] Boris Chidlovskii, Jon Ragetli, and Maarten de Rijke. Wrapper generation via grammar induction. In *ECML*, pages 96–108, 2000.

[17] Hai Leong Chieu and Hwee Tou Ng. A maximum entropy approach to information extraction from semi-structured and free text. In *Proceedings of Eighteenth national conference on Artificial intelligence*, pages 786–791, 2002.

[18] P. R. Cohen and E. A. Feigenbaum. *The Handbook of Artificial Intelligence*, volume 3. HeurisTech Press and William Kaufmann, 1982.

[19] W. Cohen, M. Hurst, and L. Jensen. *Web Document Analysis : Challenges and Opportunities*, chapter A Flexible Learning System for Wrapping Tables and Lists in HTML Documents. World Scientific, 2003.

[20] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on : http ://www.grappa.univ-lille3.fr/tata, 1997.

[21] A. Cornuéjols and L. Miclet. *Apprentissage artificiel ; concepts et algorithmes*. Eyrolles, 2002.

[22] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner : Towards automatic data extraction from large web sites. In *Proceedings of 27th International Conference on Very Large Data Bases*, pages 109–118, 2001.

[23] A. Crespo, J. Jannink, E. Neuhold, M. Rys, and R. Studer. A survey pf semi-automatic extraction and transformation. Technical report, http : //www-db.stanford.edu/ crespo/publications/extract.ps, 1994.

[24] F. DeComité, F. Denis, R. Gilleron, and F. Letouzey. Comment améliorer l'apprentissage en utilisant des exemples positifs et des exemples non étiquetés. In *CAP 99*, pages 133–144, 1999.

[25] Thomas G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7) :1895–1923, 1998.

[26] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Y. Halevy. Learning to map between ontologies on the semantic web. In *WWW*, pages 662–673, 2002.

[27] A. Finn and N. Kushmerick. Multi-level boundary classification for information extraction. In *In Proceedings of the European Conference on Machine Learning, Pisa, 2004.*, 2004.

[28] Daniela Florescu, Alon Y. Levy, and Alberto O. Mendelzon. Database techniques for the world-wide web : A survey. *SIGMOD Rec.*, 1998.

[29] Dayne Freitag and Nicholas Kushmerick. Boosted wrapper induction. In *AAAI/IAAI*, pages 577–583, 2000.

[30] Dayne Freitag and Andrew Kachites McCallum. Information extraction with hmms and shrinkage. In *Proceedings of the AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.

[31] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In Paul Vitányi, editor, *Proceedings of the 2nd European Conference on Computational Learning Theory*, volume 904 of *LNAI*, pages 23–37, Heidelberg, March 1995. Springer.

[32] Yoav Freund. Boosting a weak learning algorithm by majority. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 202–216, Rochester, New York, 6–8 August 1990. ACM Press.

[33] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Proc. 13th International Conference on Machine Learning*, pages 148–146. Morgan Kaufmann, 1996.

[34] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1) :119–139, August 1997.

[35] R. Gilleron, P. Marty, M. Tommasi, and F. Torre. Statistical classification for wrapper induction. Dagstuhl Seminar : Machine Learning for the Semantic Web, February 2005.

[36] Rémi Gilleron, Patrick Marty, Marc Tommasi, and Fabien Torre. Extraction de relations dans les documents web. In *Revue RNTI - Actes de EGC'06*, pages 415–420, 2006.

[37] Rémi Gilleron, Patrick Marty, Marc Tommasi, and Fabien Torre. Interactive tuples extraction from semi-structured data. In *2006 IEEE / WIC / ACM International Conference on Web Intelligence*, volume P2747, pages 997–1004. IEEE Comp. Soc. Press, 2006.

[38] G. Gottlob, C. Koch, R. Baumgartner, M. Herzog, and S. Flesca. The Lixto data extraction project - back and forth between theory and practice. In *23rd ACM SIGPLAN-SIGACT Symposium on Principles of Database Systems*, pages 1–12. ACM-Press, 2004.

[39] Benjamin Habegger. Tree-pattern generalization for information extraction from the web. GRAPPA Report, 2005.

[40] Benjamin Habegger and Mohamed Quafafou. Context generalization for information extraction from the web. In *ACM/IEEE Web Intelligence Conference*, pages 720–723, Beijing, China, 2004. ACM-Press.

[41] Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2) :338–355, 1984.

[42] Chun-Nan Hsu and Ming-Tzung Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Information Systems*, 23(8) :521 – 538, 1998.

[43] Lee S. Jensen and W. Cohen. Grouping extracted fields. In *Proceedings of IJCAI-2001 Workshop on Adaptive Text Extraction and Mining*, 2001.

[44] Florent Jousse. *Transformations d'Arbres XML avec des Modèles Probabilistes pour l'Annotation.* PhD thesis, Université Charles-de-Gaulle - Lille 3, October 2007.

[45] Florent Jousse, Rémi Gilleron, Isabelle Tellier, and Marc Tommasi. Champs conditionnels aléatoires pour l'annotation d'arbres. In *8ème Conférence francophone sur l'Apprentissage automatique (CAp'2006)*, pages 171–186, 2006.

[46] Florent Jousse, Rémi Gilleron, Isabelle Tellier, and Marc Tommasi. Conditional random fields for xml trees. In *ECML Workshop on Mining and Learning in Graphs*, 2006.

[47] Ron Kohavi and Clayton Kunz. Option decision trees with majority votes. In *Proc. 14th International Conference on Machine Learning*, pages 161–169. Morgan Kaufmann, 1997.

[48] Raymond Kosala, Maurice Bruynooghe, Jan Van den Bussche, and Hendrik Blockeel. Information extraction from web documents based on local unranked tree automaton inference. In *18th International Joint Conference on Artificial Intelligence*, pages 403–408. Morgan Kaufmann, 2003.

[49] Raymondus Kosala and Hendrik Blockeel. Instance-based wrapper induction. In *Benelearn 2000, Proceedings of the Tenth Belgian-Dutch Conference on Machine Learning*, pages 61–68, 2000.

[50] Trausti T. Kristjansson, Aron Culotta, Paul Viola, and Andrew McCallum. Interactive information extraction with constrained conditional random fields. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI 2004)*, 2004.

[51] N. Kushmerick. *Wrapper Induction for Information Extraction*. PhD thesis, University of Washington, 1997.

[52] Nicholas Kushmerick. Wrapper induction : Efficiency and expressiveness. *Artificial Intelligence*, 118(1-2) :15–68, 2000.

[53] Nicholas Kushmerick. Finite-state approaches to web information extraction. In *Proc. 3rd Summer Convention on Information Extraction*, 2002.

[54] A. H. F. Laender and B. Ribeiro-Neto. Debye - data extraction by example, 2001.

[55] Alberto H. F. Laender, Berthier Ribeiro-Neto, Altigran S. Silva, and Juliana S. Teixeira. A brief survey of web data extraction tools. *SIGMOD Rec.*, 31(2) :84–93, 2002.

[56] Aurélien Lemay, Joachim Niehren, and Rémi Gilleron. Learning n-ary node selecting tree transducers from completely annotated examples. In *International Colloquium on Grammatical Inference*, volume 4201 of *Lecture Notes in Artificial Intelligence*, pages 253–267. Springer Verlag, 2006.

[57] Kristina Lerman, Craig A. Knoblock, and Steven Minton. Automatic data extraction from lists and tables in web sources. In *Proceedings of Automatic Text Extraction and Mining workshop (ATEM-01), IJCAI-01*, 2001.

[58] F. Letouzey, F. Denis, and R. Gilleron. Learning from positive and unlabeled examples. In *ALT'00, Eleventh International Conference on Algorithmic Learning Theory*, Lecture Notes in Artificial Intelligence, pages 71 – 85. Springer Verlag, 2000.

[59] Ling Liu, Calton Pu, and Wei Han. XWRAP : An XML-enabled wrapper construction system for web information sources. In *ICDE*, pages 611–621, 2000.

[60] P. Marty and F. Torre. Classer pour extraire : représentations et méthodes. Technical Report Grappa report 0103, GRAPPA, december 2003.

[61] P. Marty and F. Torre. Codages et connaissances en extraction d'information. In Michel Liquière et Marc Sebban, editor, *6ième Conférence francophone sur l'apprentissage automatique*, pages 207–222. Presses Universitaires de Grenoble, 2004.

[62] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[63] Tom M. Mitchell. Generalization as search. *Artif. Intell.*, 18(2) :203–226, 1982.

[64] Stephen Muggleton and Luc De Raedt. Inductive logic programming : Theory and methods. *J. Log. Program.*, 19/20 :629–679, 1994.

[65] Ion Muslea, Steven Minton, and Craig A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2) :93–114, 2001.

[66] J. Oncina and P. García. Inference of recognizable tree sets. Technical report, Departamento de Sistemas Informáticos y Computación, Universidad de Alicante, 1993. DSIC-II/47/93.

[67] Maria Teresa Pazienza. Information extraction : Towards scalable, adaptable systems. In *Lecture Notes in Artificial Intelligence*, 1997.

[68] David Pinto, Andrew McCallum, Xing Wei, and W. Bruce Croft. Table extraction using conditional random fields. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 235–242, 2003.

[69] G. Plotkin. A note on inductive generalization. In B. Meltzer and D. Mitchie, editors, *Machine Intelligence*, volume 5, pages 153–165. Edinburgh University Press, 1970.

[70] T. Poibeau. *Extraction automatique d'information*. Hermès, Paris, 2003.

[71] J. R. Quinlan. *C4.5 : Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[72] J.R. Quinlan and R.L. Rivest. Inferring decision trees using the Minimum Description Length Principle. *Inform. Comput.*, 80(3) :227–248, March 1989. (An early version appeared as MIT LCS Technical report MIT/LCS/TM-339 (September 1987).).

[73] R. Quinlan. Data mining tools see5 and c5.0, 2004. http ://www.rulequest.com/see5-info.html.

[74] Arnaud Sahuguet and Fabien Azavant. Building intelligent web applications using lightweight wrappers. *Data Knowl. Eng.*, 36(3) :283–316, 2001.

[75] Sunita Sarawagi and William W. Cohen. Semi-markov conditional random fields for information extraction. In *Proceedings of NIPS*, pages 1185–1192, 2004.

[76] Robert E. Schapire. The boosting approach to machine learning : An overview. In *Proc. MSRI Workshop on Nonlinear Estimation and Classification*, 2002.

[77] Robert F. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the 11th Annual Conference on Computational Learning Theory (COLT-98)*, pages 80–91, New York, July 24–26 1998. ACM Press.

[78] Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3) :233–272, 1999.

[79] Jie Tang, Mingcai Hong, Juanzi Li, and Bangyong Liang. Tree-structured conditional random fields for semantic annotation. In *The Semantic Web - ISWC 2006*, volume 4273, pages 640–653, 2006.

[80] J. W. Thatcher and J. B. Wright. Generalized finite automata with an application to a decision problem of second-order logic. *Mathematical System Theory*, 2 :57–82, 1968.

[81] B. Thomas. Bottom-up learning of logic programs for information extraction from hypertext documents. In Springer-Verlag, editor, *In proceedings of European Conference on Machine Learning / Principles and Practice of Knowledge Discovery in Databases ECML/PKDD 2003*, September 2003.

[82] Bernd Thomas. *Machine Learning of Information Extraction Procedures - An ILP Approach*. PhD thesis, Universität Koblenz-Landau, 2005.

[83] Fabien Torre. Globoost : Boosting de moindres généralisés. In Michel Liquière et Marc Sebban, editor, *6ième Conférence francophone sur l'apprentissage automatique*, pages 49–64. Presses Universitaires de Grenoble, 2004.

[84] Fabien Torre. Globoost : Combinaisons de moindres généralisés. *Revue d'Intelligence Artificielle*, 19(4-5) :769–797, 2005.

[85] Geoffrey I. Webb and John W. M. Agar. Inducing diagnostic rules for glomerular disease with the DLG machine learning algorithm. *Artificial Intelligence in Medicine*, 4 :419–430, 1992.

[86] L. Zamboulis. Xml schema matching & xml data migration & integration : A step towards the semantic web vision. Viva report, Birkbeck College, 2003.

[87] Yanhong Zhai and Bing Liu. Extracting web data using instance-based learning. In *Proceedings of Web Information Systems Engineering WISE*, pages 318–331, 2005.

[88] Yanhong Zhai and Bing Liu. Web data extraction based on partial tree alignment. In *Proceedings of the 14th international conference on World Wide Web*, pages 76–85, 2005.

Table des figures

1.1 Un document non-structuré de MUC 4 9
1.2 Données à extraire du document de la figure 1.1 10
1.3 Un exemple de document HTML 11
1.4 Un exemple de document XML 13
1.5 Une partie d'une page Web du site BLS 15
1.6 Une partie du code source de la page Web de la figure 1.5 15
1.7 Représentation arborescente de la page Web de la figure 1.5 16
1.8 Un document semi-structuré : une page HTML 18
1.9 Un document XHTML contenant une liste de livres 19
1.10 Sortie structurée extrait du document XHTML de la figure 1.9 19
1.11 Une partie d'une page Web de laquelle on souhaite extraire les triplets (Club,Saison,Points
1.12 Code source HTML de la page Web de la figure 1.11 22
1.13 Requête XQUERY permettant d'extraire les triplets (Club,Saison,Date) du document HTML d
1.14 Résultat produit par l'évaluation de la requête XQUERY de la figure 1.13 sur le document HTM
1.15 Une page du site Web météorologique de la BBC 26
1.16 Résultats de ROADRUNNER 27

2.1 Moindre généralisé de deux exemples 38
2.2 Moindre généralisé d'un exemple et d'une hypothèse 38

3.1 Organisation en table : une page du site de la BBC 56
3.2 Table 57
3.3 Organisation en liste : une page du site Archidok 58
3.4 Liste 59
3.5 Organisation en table tournée : une page du site Census Bureau 61
3.6 Table tournée 62
3.7 Organisation avec factorisation 63
3.8 Organisation avec valeurs factorisées 64
3.9 Organisation en table croisée : une page du site Web Yahoo Finance . . 65
3.10 Organisation en table croisée : une page du site Web du FMI 66
3.11 Table croisée 67
3.12 Une page du site Web BEA 69
3.13 Arbre HTML d'une page du site BEA 70
3.14 Organisation tabulaire : une table 72
3.15 Organisation non tabulaire : une table tournée 72

3.16 Organisation non tabulaire : valeurs factorisées 73

4.1 Fonctionnement schématique de l'algorithme d'extraction 4 80
4.2 Trois vues d'un document arborescent 81
4.3 Codage d'un nœud 83
4.4 Codage d'une feuille 85
4.5 Codage d'une dépendance entre deux feuilles 87
4.6 Document illustrant un des effets des valeurs manquantes 91
4.7 Document illustrant le pire effet des valeurs manquantes 92

5.1 Organisation avec factorisation 119
5.2 Une portion d'arbre `HTML` 120

A.1 Exemple de document `HTML` 130

Liste des tableaux

4.1 Description statistique du corpus DataFoot 100
4.2 Résultats sur le corpus DataFoot pour dlg 101
4.3 Résultats sur le corpus DataFoot pour AdaBoostMG 101
4.4 Résultats sur le corpus DataFoot pour C5.0 102
4.5 Résultats avec l'heuristique des valeurs manquantes 102
4.6 Résultats sans l'heuristique des valeurs manquantes 103
4.7 Description statistique du corpus Rise 103
4.8 Résultats expérimentaux sur le corpus Rise 104
4.9 Comparaison des systèmes sur le corpus Rise 105
4.10 Description statistique du corpus CorporateData 105
4.11 Résultats sur le corpus CorporateData 106

5.1 Résultats des expériences interactives sur le corpus CorporateData . 124

A.1 Comparaisons des algorithmes d'induction supervisée 138